

Wireless Internet Platform for Interoperability 2.0.1

Part 3. C API

February 2004

**Korea Wireless Internet
Standardization Forum**

1.	Overview of the Required C API	2
2.	C API	4
2.1.	Kernel.....	4
2.2.	Graphics.....	67
2.3.	File	157
2.4.	Database.....	181
2.5.	Network	196
2.6.	Serial Communication	239
2.7.	User Interface Component	249
2.8.	Utility	314
2.9.	Generic I/O.....	321
2.9.1.	Overview	321
2.10.	Terminal Resource	348
2.10.1.	Overview	348
2.11.	Media Handler	391
2.12.	Placing A Call.....	459
2.13.	SMS	461
2.14.	Location Information API	463
2.14.1.	API for Location Information of a Base Station.....	463
2.14.2.	GPS Location Information API	465
2.15.	Security Communication.....	473
2.15.1.	Type of Related Data	473
2.15.2.	Related API.....	475
2.16.	Control of Supplementary Devices	491
2.17.	Mathematical Operation	501
3.	Standard C Library Function	514

1. Overview of the Required C API

This defines all APIs that can be used in Clet. Clet should implement the following functions (the platform calls the relevant function for an event when necessary):

- **handleCletEvent**

void handleCletEvent(int type, int param1, int param2)

This is a function that processes an event. An event defined in Clause 1 can be included in the type. Param1 and param2 vary depending on the value of the type.

Table 1-1. Clet event

Event (Type)	Description	Variable Used
MV_KEY_PRESS_EVENT	Event that occurs when a key is pressed	param1 = Key code value is passed param2 = 0
MV_KEY_REPEAT_EVENT	Event that occurs repeatedly at a regular time interval while a key is continuously pressed	param1 = Key code value is passed param2 = 0
MV_KEY_RELEASE_EVENT	Event that occurs when a key is released	param1 = Key code value is passed param2 = 0
MV_CHILD_APP_START_EVENT	Event indicating that a child program implemented by the current program has started	param1 = Identifier of a child program generated param2 = 0
MV_CHILD_APP_DESTROY_EVENT	Event indicating that a child program implemented by the current program has started	param1 = Identifier of a child program generated param2 = End code of the program
MV_USER_EVENT	User event	param1 = User-specified value param2 = User-specified value

- **startClet**

void startClet(int argc, char *argv[]);

This function is called when the program starts. For argc, the number of parameters is passed, and for argv, parameters. Values from argv[0] to argv[argc-1] are valid values.

- **pauseClet**

void pauseClet();

This function is called when the program stops temporarily. When it goes out of this function, the program no longer accepts events.

- **resumeClet**

void resumeClet();

This function is called when the program resumes operation. When it goes out of this function, the program accepts events again thereafter.

- **destroyClet**

void destroyClet();

Called when the program is terminated

- **paintClet**

void paintClet(int x, int y, int w, int h);

This function is called when part of the display should be painted again. X, y, w, and h represent parts of the display that should be painted.

2. C API

2.1. Kernel

- **Kernel Interface API**

Provides APIs that are related to dynamic memory allocation and release, generation and termination of a program, timer, system information, and shared memory

- **Memory Management**

Static allocation and dynamic allocation of memory are supported for the efficient management of memory. Likewise, dynamically allocated memory can support compaction. The statically allocated memory is allocated when it is loaded, and it cannot be released while the program is running. Nonetheless, it is automatically released when the program is terminated. The dynamically allocated memory by MC_knlAlloc() or MC_knlCalloc() API can be released by MC_knlFree() API while the program is running. Unreleased memory upon program termination is automatically released by the platform. Compaction occurs automatically in the absence of unused memory for allocation when dynamic memory is allocated. Dynamically allocated memory is a memory identifier (ID). Where memory compaction can occur, the user should get a pointer again from the memory ID. Compaction occurs only in the absence of unused memory for allocation when dynamic memory is allocated and when calling MC_knlGetFreeMemory() API. In case of compaction that occurs in other APIs, compaction occurs when memory is dynamically allocated within a relevant API. For APIs that dynamically allocate memory internally, refer to the section explaining side effects under each API. Among the platform APIs, some request indirect buffer as an input. These are APIs that tend to generate compaction using dynamic allocation internally.

Indirect buffer (similar to a memory ID) refers to a buffer that is allocated dynamically and a buffer that is allocated statically through DECLARE_INDIRECTBUF(). A buffer declared as DECLARE_INDIRECTBUF() has the same structure as a dynamically allocated buffer, but it is actually allocated statically.

```
typedef struct _IndirectBuf {  
    INDIRECT_BUF_HEAD;  
    char buf[1024]  
} IndirectBuf;
```

```

char staticBuf[256]; // Statically allocated buffer
char imageBuf[1024];
DECLARE_INDIRECTBUF(IndirectBuf, idBuf);

void startClet() {
    M_Byte* dynamicBuf;
    M_Uint32 mBufID;
    M_Int32 freeMemorySize;

    strcpy(staticBuf, "this is testing...\n");
    MC_knlPrintk("%s", staticBuf);

    mBufID = MC_knlCalloc(256); // mBufID is a memory ID.
    dynamicBuf = MC_GETDPTR(mBufID); // Gets a pointer from the memory ID.
    strcpy(dynamicBuf, "this is testing...\n");
    MC_knlPrintk("%s", dynamicBuf);

    freeMemorySize = MC_knlGetFreeMemory(); // Compaction occurs.
    dynamicBuf = MC_GETDPTR(mBufID);
    // Since compaction can occur, it is necessary
    // to get a pointer again from the memory ID.
    MC_knlPrintk("%s", dynamicBuf);

    MC_knlFree(mBufID);

    ...

    if ( (rID = MC_knlGetResourceID("test.gif", &rSize)) < 0 ) {
        MC_knlPrintk(resource not found\n");
        ...
    }

    mBufID = MC_knlCalloc(rSize); // Getting resource using dynamic
allocation
    MC_knlGetResource(rID, mBufID, rSize);
    dynamicBuf = MC_GETDPTR(mBufID);
    memcpy(imageBuf, dynamicBuf, rSize);

    ...
    MC_knlGetResource(rID, &idBuf, rSize);
    // Getting resource through an indirect buffer that was
allocated through DECLARE_INDIRECTBUF()
    memcpy(imageBuf, idBuf.buf, rSize);
    ...
}

```

● Program Management

The execution of multiple programs is supported. Each program has independent memory and execution space. The platform can execute multiple programs concurrently. As much as possible, however, it should not be allowed to access program resource that is not its own. In communication between programs, events and shared memory are used. A program maintains a parent-child relationship with the program that created itself. The program manager is the top-most parent where a child program is executed. A child program cannot terminate the parent program, although the parent program can terminate the child program. Likewise, terminating the parent program automatically terminates child programs.

The platform can provide an overlay function. A program developer can execute a

program that is bigger than the heap provided by the platform. To enable the overlay function, the platform should provide a development environment where the program developer can divide a large program into several small programs for compilation and bind them together as a single program for installation. In addition, when dividing a large program into several small programs, the program developer should be able to define a symbolic name for each program. The platform uses this symbolic name as the program ID when loading small programs using the overlay method. The kernel supports dynamic loading libraries. The platform enables APIs to be added to the platform as libraries and provides functions that can be shared by several programs.

- **Shared Memory**

For sharing of data among programs, shared memory is used. Shared memory can be created in duplicate using the shared memory name as a key and automatically released when all programs are terminated.

- **Others**

Several timers are supported. If the program setting the timer is terminated before the timer is released, other unreleased timers are automatically released. The kernel provides API that reads the terminal number, ESN, and other system information.

- **Buffer Allocation**

Unless otherwise specified, all APIs of the platform allocate the necessary buffer using a caller and transmit it as a parameter. A memory buffer allocated within the API that is called and returned is clearly specified in each API.

● **Additional Commands for the MC_knlGetSystemProperty Function**

Command	Remarks																						
IODEVICES	<p>A string of I/O device is supported. Several strings are separated by a comma. In the absence of a supported device, M_E_NOTSUP is returned.</p> <table border="1"> <thead> <tr> <th>String</th><th>Device</th></tr> </thead> <tbody> <tr> <td>IrDA</td><td>IrDA device</td></tr> <tr> <td>Camera</td><td>Camera device</td></tr> <tr> <td>1ChipCard</td><td>1chip IC card device</td></tr> <tr> <td>Bluetooth</td><td>Bluetooth device</td></tr> </tbody> </table> <p>When the device supports a pre-defined string, the defined string is returned. Otherwise, the string is defined and extended by the communication service provider or a vendor.</p>	String	Device	IrDA	IrDA device	Camera	Camera device	1ChipCard	1chip IC card device	Bluetooth	Bluetooth device												
String	Device																						
IrDA	IrDA device																						
Camera	Camera device																						
1ChipCard	1chip IC card device																						
Bluetooth	Bluetooth device																						
DEFAULTVOLUME	<p>A system volume category string is provided by the terminal. Several strings should be separated by a comma. In the absence of a supported system volume category, M_E_NOTSUP is returned.</p> <table border="1"> <thead> <tr> <th>String</th><th>Function</th></tr> </thead> <tbody> <tr> <td>GENERAL</td><td>General use characteristics for common applications</td></tr> <tr> <td>VOICE</td><td>Play/recording characteristics of voice</td></tr> <tr> <td>RING</td><td>This refers to the ring characteristics, e.g., when the ring is set to vibration, there will be vibration instead of sound during playing. In case of a separate speaker, sound will be generated through the speaker. In other words, when a call arrives at a terminal, set characteristics will be acted upon as they are.</td></tr> <tr> <td>KEY</td><td>Key tone characteristics</td></tr> <tr> <td>MESSAGE</td><td>Arrival alarm characteristics of an SMS message</td></tr> <tr> <td>ALARM</td><td>Alarm characteristics</td></tr> <tr> <td>ALERT</td><td>Alert characteristics, e.g., No service, Low battery, etc.</td></tr> <tr> <td>MMEDIA</td><td>Use characteristics for playing TCM2, AOD, or VOD</td></tr> <tr> <td>GAME</td><td>Play characteristics when playing a game</td></tr> <tr> <td>OEM</td><td>Used to set the sound volume that is not defined above</td></tr> </tbody> </table>	String	Function	GENERAL	General use characteristics for common applications	VOICE	Play/recording characteristics of voice	RING	This refers to the ring characteristics, e.g., when the ring is set to vibration, there will be vibration instead of sound during playing. In case of a separate speaker, sound will be generated through the speaker. In other words, when a call arrives at a terminal, set characteristics will be acted upon as they are.	KEY	Key tone characteristics	MESSAGE	Arrival alarm characteristics of an SMS message	ALARM	Alarm characteristics	ALERT	Alert characteristics, e.g., No service, Low battery, etc.	MMEDIA	Use characteristics for playing TCM2, AOD, or VOD	GAME	Play characteristics when playing a game	OEM	Used to set the sound volume that is not defined above
String	Function																						
GENERAL	General use characteristics for common applications																						
VOICE	Play/recording characteristics of voice																						
RING	This refers to the ring characteristics, e.g., when the ring is set to vibration, there will be vibration instead of sound during playing. In case of a separate speaker, sound will be generated through the speaker. In other words, when a call arrives at a terminal, set characteristics will be acted upon as they are.																						
KEY	Key tone characteristics																						
MESSAGE	Arrival alarm characteristics of an SMS message																						
ALARM	Alarm characteristics																						
ALERT	Alert characteristics, e.g., No service, Low battery, etc.																						
MMEDIA	Use characteristics for playing TCM2, AOD, or VOD																						
GAME	Play characteristics when playing a game																						
OEM	Used to set the sound volume that is not defined above																						

- **MC_PRGTYPE_JAVAAPP**

Prototype

#define MC_PRGTYPE_JAVAAPP

Description

Java application; this is defined as 1

- **MC_PRGTYPE_CAPP**

Prototype

#define MC_PRGTYPE_CAPP

Description

C application; this is defined as 2

- **MC_PRGTYPE_CDLL**

Prototype

#define MC_PRGTYPE_CDLL

Description

C dynamic loading library; this is defined as 3

- **MC_PRGTYPE_JAVADLL**

Prototype

#define MC_PRGTYPE_JAVADLL

Description

Java dynamic loading library; this is defined as 4

- **MC_PRGTYPE_JAVASYDLL**

Prototype

#define MC_PRGTYPE_JAVASYDLL

Description

Java system library; this is defined as 5

- **MC_DIR_SYS_READ_REQ_MASK**

Prototype

#define MC_DIR_SYS_READ_REQ_MASK

Description

Enables the system directory to be read ; this is defined as 0 x 01

- **MC_DIR_SYS_WRITE_REQ_MASK**

Prototype

#define MC_DIR_SYS_WRITE_REQ_MASK

Description

Enables writing on the system directory; this is defined as 0 x 02

- **MC_DIR_SHARED_READ_REQ_MASK**

Prototype

#define MC_DIR_SHARED_READ_REQ_MASK

Description

Enables reading the shared directory; this is defined as 0 x 04

- **MC_DIR_SHARED_WRITE_REQ_MASK**

Prototype

#define MC_DIR_SHARED_WRITE_REQ_MASK

Description

Enables writing on the shared directory; this is defined as 0 x 08

- **MC_NETWORK_ACCESS_REQ_MASK**

Prototype

#define MC_NETWORK_ACCESS_REQ_MASK

Description

Enables access to the network API; this is defined as 0 x 10

- **MC_SERIAL_ACCESS_REQ_MASK**

Prototype

#define MC_SERIAL_ACCESS_REQ_MASK

Description

Enables access to the serial API; this is defined as 0 x 20

- **MC_SYSTEM1_ACCESS_REQ_MASK**

Prototype

#define MC_SYSTEM1_ACCESS_REQ_MASK

Description

Enables access to APIs belonging to system group1 (defined by each mobile communication service provider); this is defined as 0 x 40

- **MC_SYSTEM2_ACCESS_REQ_MASK**

Prototype

#define MC_SYSTEM2_ACCESS_REQ_MASK

Description

Enables access APIs belonging to system group2 (defined by each mobile communication service provider); this is defined as 0 x 80

- **MC_GETDPTR**

Prototype

#define MC_GETDPTR(mID)

Description

Gets a pointer for actual use from the memory ID allocated through MC_knlAlloc() or MC_knlCalloc()

Parameters

mID Memory identifier

Return Value

Pointer

● DECLARE_INDIRECTBUF

Prototype

#define DECLARE_INDIRECTBUF(typeName, var)

Description

This function allocates an indirect buffer (memory ID) as static allocation. Allocating buffers requires defining the type of buffers to be allocated in the following formats first:

```
typedef struct type_name {  
    INDIRECT_BUF_HEAD;    // Macro provided in the platform to allocate indirect  
                           buffers  
    char buf[1024];       // Buffer setting of the size specified by the user  
};
```

Parameters

typeName	Name of the type declaration declared by the user
var	Name of the variable to be declared as an indirect buffer

Return Value

Pointer

Reference Item

For usage, refer to the example in the kernel overview.

- **MCTimer**

Prototype

typedef struct _MTimer MCTimer

Description

Structural declaration used to set a timer

Reference Item

MC_knlDefTimer, MC_knlSetTimer, MC_knlUnsetTimer

● TIMERCB**Prototype*****typedef void(*TIMERCB)(MCTimer *tm, void* parm)*****Description**

Registered in MC_knlDefTimer(), this callback function is called when the set timer expires.

Parameters

tm	Pointer of a timer structure when setting a timer
parm	Parameter transmitted when setting a timer

Reference Item

MC_knlDefTimer

- **MC_knlPrintk**

Prototype

int MC_knlPrintk(M_Char format, ...)*

Description

Prints a format string in stdout in compliance with the printf specifications of “ISO/IEC 9899:1999(E) -- Programming Languages – C.”

Side Effects

None

Reference Item

None

- **MC_knlSprintfk**

Prototype

int MC_knlSprintfk(M_Char buf, M_Char* format, ...)*

Description

Prints a format string to buf in compliance with the sprintf specifications of “ISO/IEC 9899:1999(E) -- Programming Languages – C.”

Side Effects

None

Reference Item

None

● MC_knlGetExecNames

Prototype

M_Int32 MC_knlGetExecNames(M_Char* prgName, M_Char* version, M_Char* vendor, M_Char* buf, M_Int32 bufSize)

Description

This function returns an application identification name that matches prgName (program name), version, and vendor among the applications installed in the platform. A null parameter means anything matches. For example, when prgName, version, and vendor are all null, the names of all programs installed in the platform are returned. These names are in the form of a list of strings ending with null.

Ex.) In case of two matching programs, these names can be returned while stored in a buf as shown in "/1/1.jar,3\0/2/2.jar,2\0."

Parameters

[in]	prgName	Program name
[in]	version	Version
[in]	vendor	Vendor
[out]	buf	List of strings ending with null
[in]	bufSize	Size of buf

Return Value

Pass	Relevant program count	
Fail	M_E_SHORTBUF	Issued when the buf is too small to return all names

Side Effects

None

Reference Item

None

● MC_knlExecute

Prototype

M_Int32 MC_knlExecute(M_Char* execName, M_Int32 parmCnt, ...)

Description

This function executes Java, C, or C++ program installed in the platform.

When the program has expired, or access is not allowed, an error is returned. This function is a non-blocking function. When the program is terminated, MV_CHILD_APP_DESTROY_EVENT is sent to the program manager and the parent program of the program being terminated. The executed program exists in a memory that is different from that of other programs and sends or receives data only through events and shared memory.

Transmission of parameters: A program can transmit the necessary information to a child program that it has executed, although the parameters to be transmitted should be strings.

C parent program

```
...
MC_knlExecute("execName", 4, "this parm1", "20", "this parm3", "40");
...
```

When the executed program is a C or C++ child program:

```
void startClet(int argc, char* args[]) {
    args[0];      // Program name (to be transmitted by the platform)
    args[1];      // Includes "this parm1".
    args[2];      // Includes "20".
    args[3];      // Includes "this parm3".
    args[4];      // Includes "40".
}
```

When the executed program is a Java child program:

```
public static void main(String[] args) {
    int argsLen = args.length; // The length is 5.
    args[0];      // "Program name" (to be transmitted by the platform)
    args[1];      // Includes "this parm1".
    args[2];      // Includes "20".
    args[3];      // Includes "this parm3".
    args[4];      // Includes "40".
}
```

Parameters

[in]	execName	Name of the program to be executed; it can be obtained through MC_knlGetExecNames()
[in]	parmCnt	Number of parameters to be transmitted consecutively after this parameter

Return Value

Pass

Program ID generated

Fail

M_E_ACCESS

Issued when the program has expired,
or access is not allowed

M_E_NOMEMORY

Issued in case of insufficient memory

M_E_INVALID

Issued when the parameter transmitted
is invalid**Side Effects**

None

Reference Item

None

● MC_knlExit

Prototype

void MC_knlExit(M_Int32 exitCode)

Description

This function exits the program. Actual exit time is not defined when this function is called; instead, it is defined when a return is made from this function, and it went out of the event handler. Therefore, when MC_knlExit() is called, it should be returned immediately. When the program exits through MC_knlExit(), MV_CHILD_APP_DESTROY_EVENT (exitCode is transmitted as an event parameter) is sent to the parent program and program manager. Unreleased resources when the program exits are automatically released in the platform.

The following example shows how Program A executes Program B, and how Program B exits using exitCode 27:

```

program A

void handleCletEvent(int type, int parm1, int parm2)

switch(type) {
case MV_CHILD_APP_DESTROY_EVENT:
    knlPrintk("exit code %d\n", parm1); // Prints 27
    break;

case XXX :
    ...
    execute("B", ...);
    ...
    break;

}

...
}

program B

void handleCletEvent(int type, int parm1, int parm2) {

    ...
    if ( xxx ) {
        MC_knlExit(27);
        return
    }

}

}

```

Parameters

[in]	exitCode	Exit value
------	----------	------------

Side Effects

None

Reference Item

None

- **MC_knlProgramStop**

Prototype

M_Int32 MC_knlProgramStop(M_Int32 prgID)

Description

This function stops another currently running program. A dynamic loading library cannot be terminated, however; only when all programs using the dynamic loading library are terminated will it be terminated automatically. Likewise, a parent program cannot be terminated. Only child programs can be terminated.

Parameters

[in] prgID Program ID to be terminated

Return Value

Pass

0

Fail

M_E_ACCESS

Issued when termination of a parent program is attempted

Side Effects

None

Reference Item

None

- **MC_knlGetCurProgramID**

Prototype

M_Int32 MC_knlGetCurProgramID()

Description

Gets the program ID of the currently running program

Return Value

Program ID

Side Effects

None

Reference Item

None

- **MC_knlGetParentProgramID**

Prototype

M_Int32 MC_knlGetParentProgramID()

Description

Gets the program ID of the parent program of a currently running program

Return Value

Program ID

Side Effects

None

Reference Item

None

- **MC_knlGetAppManagerID**

Prototype

M_Int32 MC_knlGetAppManagerID()

Description

Gets the program ID of the program manager

Return Value

Program ID

Side Effects

None

Reference Item

None

● MC_knlGetProgramInfo

Prototype

M_Int32 MC_knlGetProgramInfo(M_Int32* buf, M_Int32 bufSize)

Description

This function gets information on a currently operating program. The return value represents the number of currently operating programs. In a buf array, the program ID and program type come in as a pair. For example, a buf[0] of 1 and a buf[1] of MC_PRGTYPE_JAVADLL mean that the type of a program whose program ID is 1 is java application DLL. Therefore, the size of an array is two times the number of programs.

Parameters

[out]	buf	Buffer to which a program type obtained is returned
[in]	bufSize	Buffer size to be transmitted

Return Value

Pass	Number of currently running programs	
Fail	M_E_SHORTBUF	Issued when the buffer is too small to return all names

Side Effects

None

Reference Item

None

- **MC_knlGetAccessLevel**

Prototype

M_Int32 MC_knlGetAccessLevel()

Description

This function gets the access level of a program. Each bit of the return value represents the types of APIs that can be accessed by the current program. The meaning of each bit depends on the mask values of MC_DIR_SYS_READ_REQ_MASK, MC_DIR_SYS_WRITE_REQ_MASK, MC_DIR_SHARED_READ_REQ_MASK, MC_DIR_SHARED_WRITE_REQ_MASK, MC_NETWORK_ACCESS_REQ_MASK, MC_SERIAL_ACCESS_REQ_MASK, MC_SYSTEM1_ACCESS_REQ_MASK, MC_SYSTEM2_ACCESS_REQ_MASK.

Return Value

Access level (OR value of each mask value)

Side Effects

None

Reference Item

None

- **MC_knlGetProgramName**

Prototype

M_Int32 MC_knlGetProgramName(M_Char nameBuf, M_Int32 bufSize)*

Description

Gets the program name of the currently running program; the name to be obtained is the one described in the ADF file

Parameters

[out]	nameBuf	Buffer to which the name obtained is returned
[in]	bufSize	namebuf size

Return Value

Pass	0	
Fail	M_E_SHORTBUF	Issued when the buffer is too small to return all names

Side Effects

None

Reference Item

None

- **MC_knlCreateSharedBuf**

Prototype

M_Uint32 MC_knlCreateSharedBuf(const M_Char* name, M_Int32 size)

Description

This function creates a shared buffer. A shared buffer enables data sharing among different programs. When all the programs using this buffer are terminated, the buffer created is automatically removed.

Parameters

[in]	name	Name of the shared buffer (ends with null)
[in]	size	Size of the buffer to be created

Return Value

Pass	Indirect buffer (memory ID) created
Fail	In case of a shared buffer having the same name, or when memory is insufficient

Side Effects

Compaction may occur in case of insufficient memory when creating a shared buffer.

Reference Item

None

● MC_knlDestroyShareBuf

Prototype

M_Int32 MC_knlDestroySharedBuf(M_Uint32 buf)

Description

Destroys the shared buffer created

Parameters

[in] buf Shared indirect buffer (memory ID)

Return Value

Pass

0

Fail

M_E_INVALID

Issued in the absence of the memory ID of the shared buffer

M_E_ERROR

Issued when the shared buffer cannot be destroyed due to other reasons

Side Effects

None

Reference Item

None

- **MC_knlGetSharedBuf**

Prototype

M_Uint32 MC_knlGetSharedBuf(const M_char* name)

Description

Gets a shared buffer that is created with a name string

Parameters

[in] name Name of the shared buffer (ends with null)

Return Value

Pass

Shared indirect buffer (memory ID)

Fail

0 In the absence of a set shared buffer

Side Effects

None

Reference Item

None

- **MC_knlGetSharedBufSize**

Prototype

M_Int32 MC_knlGetSharedBufSize(M_Uint32 buf)

Description

Gets the size of a shared buffer

Parameters

[in]	buf	Shared indirect buffer (memory ID); this is the return value of the MC_knlCreateSharedBuf() or MC_knlGetSharedBuf() function
------	-----	--

Return Value

Pass	Size of the shared buffer
Fail	M_E_NOTEXIST Issued in the absence of a set shared buffer

Side Effects

None

Reference Item

None

- **MC_knlResizeSharedBuf**

Prototype

M_Uint32 MC_knlResizeSharedBuf(M_Uint32 buf, M_Int32 size)

Description

Changes the size of a shared buffer

Parameters

[in]	buf	Shared indirect buffer (memory ID)
[in]	size	Size of a shared buffer to be created

Return Value

Pass	Shared indirect buffer (memory ID) whose size has been changed
Fail	0

Side Effects

Compaction may occur in case of insufficient memory when changing the size of a shared buffer.

Reference Item

None

- **MC_knlAlloc**

Prototype

M_Uint32 MC_knlAlloc(M_Int32 size)

Description

This function allocates as much memory as required by a heap. The allocated memory can be re-used only when it is released by MC_knlFree(). When the allocated memory is not released until the program is terminated, it is automatically released in the platform upon program termination. When used as a memory ID, the allocated memory should get a pointer using MC_GETDPTR().

Parameters

[in]	size	Size of the allocation requested (byte)
------	------	---

Return Value

Pass	Memory ID allocated
Fail	0

Side Effects

Compaction may occur in case of insufficient memory.

Reference Item

None

- **MC_knlCalloc**

Prototype

M_Uint32 MC_knlCalloc(M_Int32 size)

Description

This function allocates as much memory as required by a heap. The allocated memory can be re-used only when it is released by MC_knlFree(). When the allocated memory is not released until the program is terminated, it is automatically released in the platform upon program termination. When used as a memory ID, the allocated memory should get a pointer using MC_GETDPTR().

Parameters

[in]	size	Size of the allocation requested (byte)
------	------	---

Return Value

Pass	Memory ID allocated
Fail	0

Side Effects

Compaction may occur in case of insufficient memory.

Reference Item

None

- **MC_knlFree**

Prototype

void MC_knlFree(M_Uint32 mID)

Description

Releases the memory allocated through MC_knlCalloc(); releasing an unallocated memory should not be attempted

Parameters

[in] mID Memory ID to be released

Side Effects

None

Reference Item

None

- **MC_knlGetTotalMemory**

Prototype

M_Int32 MC_knlGetTotalMemory()

Description

Gets the total memory

Return Value

Total memory (byte)

Side Effects

None

Reference Item

None

- **MC_knlGetFreeMemory**

Prototype

M_Int32 MC_knlGetFreeMemory()

Description

This function carries out compaction and gets and returns free memory in case memory compaction is explicitly required.

Return Value

Free memory (byte)

Side Effects

None

Reference Item

None

- **MC_knlDefTimer**

Prototype

void MC_knlDefTimer(MCTimer* tm, TIMERCB cb)

Description

This function initializes the timer and registers the callback function.

When the timer expires, the registered callback function is called.

When calling the callback function, the pointer of the timer structure that was set in MC_knlSetTimer() and parameters are transmitted.

```
MCTimer tm;

void TimerCb(MCTimer* ptm, void* parm) {
    MC_knlPrintk("timer occur %d\n", parm);
    MC_knlSetTimer(ptm, 1000L, (int)parm+1);
}

void startClet(int argc, char* argv[]) {
    MC_knlPrintk("start Clet!!!\n");
    MC_knlDefTimer(&tm, TimerCb);
    MC_knlSetTimer(&tm, 1000L, 0x1234);
}
```

Parameters

[in]	tm	Pointer of the timer structure to be initialized
[in]	cb	Timer callback function

Side Effects

None

Reference Item

None

● MC_knlSetTimer

Prototype

M_Int32 MC_knlSetTimer(MCTimer* tm, M_Int64 timeout, void* parm)

Description

This function sets the timer. When the timer expires, the callback function set by MC_knlDefTimer() is called. When the program is terminated while the timer has not yet expired, the remaining timers are automatically released.

An error occurs if an unexpired timer is re-defined using this function. Re-defining an unexpired timer requires calling MC_knlUnsetTimer() first before calling this function.

In case another task is being implemented, an error may occur between the timer and the timer resolution supported by the operating system at the bottom of the platform when the timer expires.

Parameters

[in]	tm	Pointer of the timer structure
[in]	timeout	In millisecond
[in]	parm	Parameter that will be transmitted to the callback function when the timer expires

Return Value

Pass	0	
Fail	M_E_EXIST	When the existing timer has not yet expired

Side Effects

None

Reference Item

None

- **MC_knlUnsetTimer**

Prototype

void MC_knlUnsetTimer(MCTimer* tm)

Description

Cancels a set timer; this function is ignored when no timer is set

Parameters

[in] tm Pointer of the timer structure

Side Effects

None

Reference Item

None

- **MC_knlCurrentTime**

Prototype

M_Int64 MC_knlCurrentTime()

Description

Gets the current time in millisecond

Return Value

Milliseconds calculated from 0 hour 0 minute 0 second (00:00:0), January 1, 1970
to the present time

Side Effects

None

Reference Item

None

● MC_knlGetSystemProperty

Prototype

```
M_Int32 MC_knlGetSystemProperty(M_Char* id, M_Char* rtnBuf,
M_Int32 bufSize)
```

Description

This function reads a specified value on the terminal. The ID string that can be used as a parameter should comply with the string used in MH_sysGetInformation() of HAL document API. Depending on the mobile communication service provider or vendor, the ID string can be further extended.

Strings such as "ESN," "NID," "SID," "BASELAT," "BASELONG," "CURRENTCH," "PHONENUMBER," "RSSILEVEL," "BATTERYLEVEL," "DATASERVICEMODE," "MAXSOCKETNUM," "MAXRSSILEVEL," "MAXSERIALNUM," "MAXBATTLEVEL," "MEDIADVICES," "DNS," "VIBRATORLEVEL", "VOLUMELEVEL", "IODEVICES", and "DEFAULTVOLUME" can be used. The id string may vary depending on the terminal mounted.

Parameters

[in]	id	String to be read
[out]	rtnBuf	Buffer to which the return string is returned
[in]	bufSize	Size of the buffer where the return value is stored

Return Value

Pass	0	
Fail	M_E_SHORTBUF	Issued when the size of the buffer transmitted is smaller than the string to be returned
	M_E_INVALID	Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

None

- **MC_knlSetSystemProperty**

Prototype

M_Int32 MC_knlSetSystemProperty(M_Char* id, M_Char* buf)

Description

This function sets a specified value on the terminal. The ID string that can be used as parameter and buf string should comply with the string used in MH_sysSetInformation() of HAL document API. Depending on the mobile communication service provider or vendor, the ID string can be further extended.

Parameters

[in]	id	ID string to be set
[in]	buf	String buffer to be set

Return Value

Pass	0	
Fail	M_E_INVALID	Issued when the transmitted parameter is invalid
	M_E_ACCESS	Issued when the information cannot be set

Side Effects

None

Reference Item

None

- **MC_knlGetResourceID**

Prototype

M_Int32 MC_knlGetResourceID(M_Char resourceName, M_Int32* size)*

Description

Gets the resource ID related to the program

Parameters

[in]	resourceName	Resource name
[out]	size	Resource size

Return Value

Pass	Resource ID
Fail	M_E_NOENT Issued when the resource does not exist

Side Effects

None

Reference Item

None

● MC_knlGetResource

Prototype

M_Int32 MC_knlGetResource(M_Int32 resourceID, M_Uint32 buf, M_Int32 bufSize)

Description

Reads the resource corresponding to the resource ID

Parameters

[in]	resourceID	Resource ID
[in]	buf	Indirect buffer (memory ID) where the resource is copied
[in]	bufSize	Buffer size

Return Value

Pass

0

Fail

M_E_SHORTBUF

Issued when the buffer size is too small

Side Effects

Compaction may occur.

Reference Item

None

● MC_DLL_INTERFACE

Prototype

MC_DLL_INTERFACE(void* interface, M_Char* interfaceName, M_Int32 major, M_Int32 minor)

Description

This function defines the entry of an interface to be exported. The entry defined here is used as a parameter for MC_EXPORT_DLL_INTERFACE_LIST(). Values assigned to interfaceName, major, and minor will be used as targets when MC_knlGetDLLInterface() searches a matching interface.

Parameters

[in]	interface	Interface defined by the program developer
[in]	interfaceName	Name to be assigned to the interface
[in]	major	Major version number to be assigned to the interface
[in]	minor	Minor version number to be assigned to the interface

- **MC_EXPORT_DLL_INTERFACE_START(dllName),
MC_EXPORT_DLL_INTERFACE_END**

Prototype

MC_EXPORT_DLL_INTERFACE_START(dllName)

MC_DLL_INTERFACE()

MC_DLL_INTERFACE()

...

MC_DLL_INTERFACE()

MC_EXPORT_DLL_INTERFACE_END

Description

Defines the list of interfaces to be exported

Parameters

[in] dllName	DLL name of string type (e.g., MyNetworkDLL)
--------------	--

- **MC_DLL_INIT**

Prototype***MC_DLL_INIT(M_Int32 (*initFunc)(void))*****Description**

The function that should be called when DLL is loaded is defined. Using the defined function, the DLL developer can implement the required initialization before the DLL can be used. When this function is not declared, the initialization function will not be called when DLL is loaded. A value of 0 should be returned when M_Int32 (*initFunc)(void) operates normally. When a negative value is returned, the corresponding DLL is not loaded in MC_knlLoad(); instead, it returns M_E_INIT.

- **MC_DLL_EXIT**

Prototype***MC_DLL_EXIT(void (*exitFunc)(void))*****Description**

The function that should be called when DLL is released from memory is defined. Using the defined function, the DLL developer can take the necessary action (free resource, etc.) before DLL is terminated. When this function is not declared, the exit function is not called during unloading.

- **MC_EXPORT_DLL_START(dllName), MC_EXPORT_DLL_END**

Prototype***MC_EXPORT_DLL_START(dllName)******MC_DLL_INIT()******MC_DLL_EXIT()******MC_EXPORT_DLL_END*****Description**

This is the function that is called first when calling DLL, and when DLL is released from memory. dllName acts as a separator when several DLL modules are used for one image as built-in dll. Likewise, MC_EXPORT_DLL_INTERFACE_START and MC_EXPORT_DLL_START should be in the same file using the same dllName.

Example 1:

```
MC_EXPORT_DLL_INTERFACE_START(test1)
```

```
MC_EXPORT_DLL_INTERFACE_END
```

```
MC_EXPORT_DLL_START(test1)
```

```
MC_EXPORT_DLL_END
```

Example 2:

```
MC_EXPORT_DLL_INTERFACE_START(test2)
```

```
    MC_DLL_INTERFACE(interface1, "testInterface1", 1, 0)
```

```
MC_EXPORT_DLL_INTERFACE_END
```

```
MC_EXPORT_DLL_START(test2)
```

```
MC_EXPORT_DLL_END
```

```
MC_EXPORT_DLL_INTERFACE_START(test3)
```

```
    MC_DLL_INTERFACE(interface2, "testInterface2", 1, 0)
```

```
MC_EXPORT_DLL_INTERFACE_END
```

```
MC_EXPORT_DLL_START(test3)
```

```
    MC_DLL_INIT(initFunc)
```

```
MC_EXPORT_DLL_END
```

Parameters

[in] dllName	DLL name of string type (e.g., MyNetworkDLL)
--------------	--

● MC_knlLoad

Prototype

M_Int32 MC_knlLoad(char* dllLibName, int parmCnt, ...)

Description

This function loads the dynamic linking library (DLL) set in the platform.

When the DLL has expired, or access is not allowed, an error is returned. This function is a non-blocking function. When the program loads the library, the two are in the same memory space, thereby enabling the library function to be called for immediate use. When different programs load the same library, the library can be shared according to the value (i.e., ADF) set at the time of library registration or loaded separately. The library is automatically terminated when all programs that loaded the library explicitly implement MC_knlUnload(), or when all programs that are currently in use are terminated.

Parameters

[in]	dllLibName	DLL name to be mounted; this can be obtained using MC_knlGetExecNames()
[in]	parmCnt	Number of parameters to be transmitted consecutively after this parameter

Return Value

Pass

ID of the program generated

Fail

M_E_ACCESS	Issued when the program has expired, or access is not allowed
M_E_NOMEMORY	Issued in case of insufficient memory
M_E_INVALID	Issued when the parameter transmitted is invalid
M_E_INIT	Issued in case of an error during the initialization of DLL loading

Side Effects

None

Reference Item

None

● MC_knlUnload

Prototype

M_Int32 MC_knlUnload(M_Int32 dllID)

Description

This function disconnects the link with the dynamic linking library (DLL) loaded in the platform.

In case different programs loaded the same library, actual termination occurs through MC_knlUnload(), which is finally executed when all loaded programs execute the MC_knlUnload() function. When the program that loaded the library is terminated without executing MC_knlUnload(), actual termination occurs automatically upon program termination.

Parameters

[in] dllID ID obtained using MC_knlLoad()

Return Value

Pass

0

Fail

M_E_INVALID

Issued when the parameter transmitted is invalid

Side Effects

None

Reference Item

None

● MC_knlGetDLLInterface

Prototype

```
void* MC_knlGetDLLInterface(M_Char* name, M_Int32 major, M_Int32  
minor, M_Int32* rtnMajor, M_Int32* rtnMinor);
```

Description

This function gets an interface from the loaded dynamic linking library (DLL).

The current program finds and returns an interface that matches the name, major, and minor from among DLLs loaded with MC_knlLoad() and built-in DLLs. In case of interfaces that match those from the libraries loaded during runtime and built-in libraries, the libraries loaded during runtime should be checked first. When there are more than one interface matching those from among the libraries loaded during runtime, the most recently loaded one should be checked first. An interface can be loaded not only through MC_knlLoad() but also through built-in, APM automatic loading and manipulation by the program developer. In such cases, DLLs that were not loaded through MC_knlLoad() can be found through MC_knlGetDLLInterface().

The following are the matching conditions:

1. (Name matches.) && (target interface major > parameter major) :: (minor version is ignored)
2. (Name matches.) && (target interface major == parameter major) && (target interface minor >= parameter minor)

Parameters

[in]	name	Interface name
[in]	major	Interface major version
[in]	minor	Interface minor version
[out]	rtnMajor	Major version of the interface that is actually found; in case a return value is not necessary, null is transmitted
[out]	rtnMinor	Minor version of the interface that is actually found; in case a return value is not necessary, null is transmitted

Return Value

Pass	Interface pointer
Fail	NULL
	Issued when a matching interface cannot be found

Side Effects

None

Reference Item

None

2.2. Graphics

- **Graphic Function**

APIs that can be used to paint on the display or off-screen frame buffers

- **Frame Buffer**

Display frame buffers and off-screen frame buffers are described in data pointer and byte per line. Such frame buffers are defined through MC_GrpFrameBuffer, which contains data, byte per line for data, width, and height.

- **Generation/Extinction of Off-Screen Frame Buffer**

A frame buffer is created through the MC_grpCreateOffScreenFrameBuffer function. It can also get a display frame buffer that exists through the MC_grpGetScreenFrameBuffer function. The frame buffer created should be removed using the MC_grpDestroyScreenFrameBuffer function.

- **Graphic Context**

Through a function that paints on a frame buffer, most of the graphic contexts are passed on as parameters. The graphic context is a structure that stores various parameters that are necessary for painting, such as foreground color, clipping area, degree of transparency, pixel operation function, font and style, and offset for relative coordinates.

To change the value for each graphic context, the MC_grpSetContext function is used. Painting is done when a clipping area is specified for all painting APIs. Depending on the API, each field value may or may not be used.

The painting mode includes transparent painting (alpha mode), XOR painting (xor mode), and common mode (none of the above). Each of these modes is an exclusive mode, i.e., if the alpha mode is specified in the xor mode, the previous xor mode is canceled.

When a relative coordinate system is specified, all painting coordinates are changed based on such relative coordinate system.

- **Image Decoding/Encoding**

An image consists of frame buffer, state of animation, and mask image.

The image format supports BMP and PNG. Image content can be printed on the display when, after loading the content of an original image to the buffer, an image structure is created using the MC_grpCreateImage function. An image can be printed on the display through the MC_grpDrawImage function. In case of an animated image, the next image can be created using the MC_grpDecodeNextImage function. Used images are removed using the MC_grpDestroyImage function.

With the MC_grpEncodeImage function, the desired frame buffer area can be encoded in BMP.

- **Double Buffering**

In most cases, the LCD of a phone is separated from the host memory, and updating the display content takes some time. To address this problem, the double buffering technique is used. Even when something is drawn on the display frame buffer, it is not actually printed on the display. Only when the MC_grpFlushLcd function is called is the content of the display frame buffer printed on the LCD display.

- **Dual Display**

There can be two LCDs depending on the phone. Likewise, the supplementary LCD may vary from the main LCD in terms of the number of colors provided. Even when the number of colors is different, the color of the frame buffer received through the MC_grpFlushLcd function uses the same format as that of the main LCD (the number of colors is also the same). MC_grpFlushLcd a color that is closest to that color.

- **Handling of Character Input**

Input keys are received from the user, and their values are checked. Characters are assembled through an automata and passed through a completed string buffer and a string buffer being assembled. The automata determines whether a string can be inputted, and the currently available strings in the automata can be obtained through MC_imGetSupportModeCount(). The type that is returned is a string pointer, and the data should comply with the standard language code defined in the ISO 639 code. In addition, in case the language differentiates the upper case character and lower case character, "/S" and "/L" can be added to each code. For the language code for the number input, "N123" is used.

- **MC_GrpPixelOpProc**

Prototype

```
typedef M_Int32(*MC_GrpPixelOpProc)(M_Int32 srcpxl, M_Int32 orgpxl,  
M_Int32 param1)
```

Description

This is a pixel operation function. In all painting functions, the result value of this function becomes the value that is finally used. When srcpxl and orgpxl are calculated properly as parameters of param1, and the result value is returned, the value becomes the final value that is used on the frame buffer. For example, a function that is used in transparent mode should be constructed as follows:

Parameters

[in]	srcpxl	Pixel value of the frame buffer
[in]	destpxl	Pixel value to be used for the frame buffer
[in]	param1	Parameters that control the function

Return Value

Pixel value to be used finally for the frame buffer

```
int alphaOp(int srcpxl, int orgpxl, int param1){  
    return (srcpxl * (255 - param1) + orgpxl * param1) / 255;  
}
```

Side Effects

None

Reference Item

None

- **MC_GrpContext**

Prototype

typedef struct _MC_GrpContext MC_GrpContext

Description

Graphic context

This is a structure that is used to transmit various parameters for painting effectively. Most of the painting functions receive this structure as parameters. Included in this structure are parameters required for painting such as clipping, foreground color, font, style, and relative coordinate system.

- **MC_GRP_DIRECT_COLOR_TYPE**

Prototype

#define MC_GRP_DIRECT_COLOR_TYPE

Description

Color type that does not use a palette; this is defined as 1

- **MC_GRP_GRAY_TYPE**

Prototype

#define MC_GRP_GRAY_TYPE

Description

Black and white type; this is defined as 2

- **MC_GRP_COLOR_TYPE**

Prototype

#define MC_GRP_COLOR_TYPE

Description

Color type; this is defined as 4

- **MC_GrpDisplayInfo**

Prototype***struct _MC_GrpDisplayInfo***

```
int m_bpp           // Number of bits per pixel
int m_depth        // Number of bits actually used per pixel
int m_width         // Width of the display in pixel
int m_height        // Height of the display in pixel
int m_bpl           // Number of bytes per line of the frame buffer on
                        the display
int m_colortype      //Color type; MC_GRP_DIRECT_COLOR_TYPE,
                        MC_GRP_GRAY_TYPE, MC_GRP_COLOR_TYPE
int m_redmask        // Red mask
int m_bluemask       // Blue mask
int m_greenmask      // Green mask
```

Description

Displays the information structure

- **MC_GrpFrameBuffer**

Prototype

typedef M_Int32 MC_GrpFrameBuffer

Description

Frame buffer

Internally, the frame buffer has height, width, and frame buffer pointer.

● MC_GRP_GET_FRAME_BUFFER_POINTER**Prototype*****#define MC_GRP_GET_FRAME_BUFFER_POINTER(a)*****Description**

Returns the pointer of the frame buffer; in case NULL is returned, direct access using the application pointer is not approved

Parameters

[in] a MC_GrpFrameBuffer

Return Value

In case direct access is approved, access the pointer featuring the contents of the frame buffer. Otherwise, in case of non-approval, NULL is returned.

Side Effects

A screen frame buffer pointer obtained by MC_grpGetScreenFrameBuffer returns NULL.

- **MC_GRP_GET_FRAME_BUFFER_WIDTH**

Prototype

```
#define MC_GRP_GET_FRAME_BUFFER_WIDTH(a)
```

Description

Returns the width of the frame buffer

Parameters

[in] a MC_GrpFrameBuffer

Return Value

Frame buffer pointer

- **MC_GRP_GET_FRAME_BUFFER_HEIGHT**

Prototype

#define MC_GRP_GET_FRAME_BUFFER_HEIGHT(a)

Description

Returns the height of the frame buffer

Parameters

[in] a MC_GrpFrameBuffer

Return Value

Height of the frame buffer

- **MC_GRP_GET_FRAME_BUFFER_BPL**

Prototype***#define MC_GRP_GET_FRAME_BUFFER_BPL(a)*****Description**

Returns the number of bytes per line of the frame buffer

Parameters

[in] a MC_GrpFrameBuffer

Return Value

Number of bytes per line of the frame buffer

- **MC_GRP_GET_FRAME_BUFFER_BPP**

Prototype

```
#define MC_GRP_GET_FRAME_BUFFER_BPP(a)
```

Description

Returns the number of bits per pixel of the frame

Parameters

[in] a MC_GrpFrameBuffer

Return Value

Number of bits per pixel

- **MC_GRP_CONTEXT_CLIP_IDX**

Prototype***#define MC_GRP_CONTEXT_CLIP_IDX*****Description**

This function specifies a rectangle that indicates a clipping area.

A rectangle is described by a dot on the upper left and a dot on the lower right. The dot on the upper left is included in the rectangle, but not the dot on the lower right. This is defined as 0.

- **MC_GRP_CONTEXT_FG_PIXEL_IDX**

Prototype

#define MC_GRP_CONTEXT_FG_PIXEL_IDX

Description

Specifies the value for a foreground color pixel; this is defined as 1

- **MC_GRP_CONTEXT_BG_PIXEL_IDX**

Prototype

#define MC_GRP_CONTEXT_BG_PIXEL_IDX

Description

Specifies the value for a background color pixel; this is defined as 2

- **MC_GRP_CONTEXT_ALPHA_IDX**

Prototype***#define MC_GRP_CONTEXT_ALPHA_IDX*****Description**

This function specifies the degree of transparency for painting.

A transparency of 0 means that the painting does not show on the display. When the transparency is 255, however, the painting is shown on the display, albeit not transparently. This is defined as 4.

- **MC_GRP_CONTEXT_PIXELOP_IDX**

Prototype

#define MC_GRP_CONTEXT_PIXELOP_IDX

Description

Specifies a pixel operation function; this is defined as 5

Reference Item

MC_GrpPixelOpFunc

- **MC_GRP_CONTEXT_PIXEL_PARAM1_IDX**

Prototype***#define MC_GRP_CONTEXT_PIXEL_PARAM1_IDX*****Description**

This function specifies the parameters for a pixel operation function.

The third parameter that is passed when the pixel operation function is called is specified. This is defined as 6.

- **MC_GRP_CONTEXT_FONT_IDX**

Prototype

#define MC_GRP_CONTEXT_FONT_IDX

Description

Specifies the font ID obtained through the MC_getFont function; this is defined as 7

- **MC_GRP_CONTEXT_STYLE_IDX**

Prototype***#define MC_GRP_CONTEXT_STYLE_IDX*****Description**

This function specifies the line drawing style. Either MC_GRP_SOLID_STYLE or MC_GRP_DOTTED_STYLE is specified. This is defined as 8.

- **MC_GRP_CONTEXT_XOR_MODE_IDX**

Prototype***#define MC_GRP_CONTEXT_XOR_MODE_IDX*****Description**

This function specifies the drawing mode.

In particular, it specifies whether or not XOR will be used for drawing. A value of 1 means that the drawing will be in XOR; otherwise, the drawing will be in common mode.

This is defined as 9.

- **MC_GRP_CONTEXT_OFFSET_IDX**

Prototype***#define MC_GRP_CONTEXT_OFFSET_IDX*****Description**

This function specifies the offset of relative coordinates for drawing.

The dot coordinates of relative coordinates are specified in an integer array. This is defined as 10.

- **MC_GRP_SOLID_STYLE**

Prototype

#define MC_GRP_SOLID_STYLE

Description

Uses an identical color when drawing a line; this is defined as MG_FB_SOLID_STYLE

- **MC_GRP_DOTTED_STYLE**

Prototype

#define MC_GRP_DOTTED_STYLE

Description

Draws every other dot when drawing a line; this is defined as MG_FB_DOTTED_STYLE

- **MC_GRP_FT_SIZE_SMALL**

Prototype

#define MC_GRP_FT_SIZE_SMALL

Description

Specifies the size of a small font; this is defined as 8

- **MC_GRP_FT_SIZE_MEDIUM**

Prototype

#define MC_GRP_FT_SIZE_MEDIUM

Description

Specifies the size of a medium-sized font; this is defined as 0

- **MC_GRP_FT_SIZE_LARGE**

Prototype

#define MC_GRP_FT_SIZE_LARGE

Description

Specifies the size of a large font; this is defined as 16

- **MC_GRP_FT_FACE_SYSTEM**

Prototype

#define MC_GRP_FT_FACE_SYSTEM

Description

Specifies the font face that is used in the system; this is defined as 0

- **MC_GRP_FT_FACE_MONOSPACE**

Prototype

#define MC_GRP_FT_FACE_MONOSPACE

Description

Specifies a monospaced font face; this is defined as 32

- **MC_GRP_FT_FACE_PROPORTIONAL**

Prototype

#define MC_GRP_FT_FACE_PROPORTIONAL

Description

Specifies a font face that is not monospaced; this is defined as 64

- **MC_GRP_FT_STYLE_PLAIN**

Prototype

#define MC_GRP_FT_STYLE_PLAIN

Description

Specifies a plain font style; this is defined as 0

- **MC_GRP_FT_STYLE_BOLD**

Prototype

#define MC_GRP_FT_STYLE_BOLD

Description

Specifies a bold font style; this is defined as 1

- **MC_GRP_FT_STYLE_ITALIC**

Prototype

#define MC_GRP_FT_STYLE_ITALIC

Description

Specifies an italic font style; this is defined as 2

- **MC_GRP_FT_STYLE_UNDERLINE**

Prototype

#define MC_GRP_FT_STYLE_UNDERLINE

Description

Specifies an underlined font style; this is defined as 4

- **MC_GRP_IMAGE_DONE**

Prototype

#define MC_GRP_IMAGE_DONE

Description

Notifies that the entire image source decoding is completed; this is defined as 1

- **MC_GRP_FRAME_DONE**

Prototype***#define MC_GRP_FRAME_DONE*****Description**

Indicates that the image of one frame from the image source is completed; this is defined as 0

- **MC_GrplImage**

Prototype

typedef void * MC_GrplImage

Description

An image has a frame buffer and other properties (including animation status, etc.) within itself.

- **MC_GRP_IS_ANIMATED**

Prototype

#define MC_GRP_IS_ANIMATED

Description

An image property indicating the animation status; this is defined as 1

- **MC_GRP_ANIMATE_DELAY**

Prototype

#define MC_GRP_ANIMATE_DELAY

Description

An image property indicating the animation delay; this is defined as 2

- **MC_GRP_LOOP_COUNT**

Prototype

#define MC_GRP_LOOP_COUNT

Description

Property of the animation loop count image; this is defined as 3

● MC_grpGetImageProperty**Prototype**

int MC_grpGetImageProperty(MC_GrpImage img, M_Int32 index)

Description

Returns the image property

MC_GRP_IS_ANIMATED	Animation status: Animation when 1; non-animation when 0
MC_GRP_ANIMATE_DELAY	Animation delay unit: Millisecond
MC_GRP_LOOP_COUNT	Entire animation loop count
MC_GRP_IMAGE_WIDTH	Width of the image
MC_GRP_IMAGE_HEIGHT	Height of the image
MC_GRP_IMAGE_BPP	Number of bits per pixel
MC_GRP_CURRENT_FRAME	Current frame index of the animation; the first frame is 0

Parameters

img Image

index Image property

Return Value

Property value

Side Effects

None

Reference Item

None

- **MC_grpGetImageFrameBuffer**

Prototype

MC_GrpFrameBuffer MC_grpGetImageFrameBuffer(MC_GrpImage img)

Description

Returns the frame buffer of an image

Parameters

img Image

Return Value

Frame buffer of the image

Side Effects

None

Reference Item

None

- **MC_grpGetScreenFrameBuffer**

Prototype

MC_GrpFrameBuffer MC_grpGetScreenFrameBuffer(M_Int32 i)

Description

Gets the frame buffer of the current screen (corresponds to each I)

Parameters

[in]	I	Frame buffer of the main LCD display when the frame buffer of the screen is 0; frame buffer of the external auxiliary LCD display when the frame buffer of the screen is 1
------	---	--

Return Value

Frame buffer corresponding to the screen; in the absence of a screen frame buffer that corresponds to I, null is returned

Side Effects

None

Reference Item

None

- **MC_grpDestroyOffScreenFrameBuffer**

Prototype

void MC_grpDestroyOffScreenFrameBuffer(MC_GrpFrameBuffer fb)

Description

Destroys the offscreen frame buffer created; when the screen frame buffer is regarded as a parameter, no action is taken

Parameters

[in] fb Offscreen frame buffer to be destroyed

Side Effects

None

Reference Item

None

- **MC_grpCreateOffScreenFrameBuffer**

Prototype

MC_GrpFrameBuffer ***MC_grpCreateOffScreenFrameBuffer(M_Int32 w, M_Int32 h)***

Description

This function creates an offscreen frame buffer with specified width and height in the memory. The created offscreen frame buffer can have the same number of colors as the main screen LCD.

w and h should be numbers that are larger than 0.

Parameters

[in]	w	Width of the frame buffer
[in]	h	Height of the frame buffer

Return Value

New frame buffer created; when w or h is smaller than 0, or in case of insufficient memory when the frame buffer is created, null is returned

Side Effects

None

Reference Item

None

- **MC_grpInitContext**

Prototype

void MC_grpInitContext(MC_GrpContext* pgc)

Description

This function initializes all the fields of the graphic context. Font, offset of the relative coordinates, color, stroke style, alpha value, XORMode, and clipping area are all initialized to default values.

Specifically, the font becomes the default system font, color becomes black, painting mode becomes plain mode, clipping area is released, and offset of the relative coordinates becomes (0, 0).

Parameters

[in] pgc Pointer of MC_GrpContext to be initialized

Side Effects

None

Reference Item

None

● MC_grpSetContext

Prototype

void MC_grpSetContext(MC_GrpContext* pgc, M_Int32 index, void* pv)

Description

Changes a specific part of the graphic context; content to be placed in pv varies depending on the index value

idx value	pv value
MC_GRP_CONTEXT_CLIP_IDX	As a clipping area, an array of integers is placed in pv, and this array of integers is copied. The x and y coordinates of the upper left of the clipping rectangle are placed in the first and second positions, respectively. Similarly, the x and y coordinates of the lower right of the clipping rectangle are placed in the third and fourth positions, respectively. In this case, a dot on the upper left is included in the clipping area, but not a dot on the lower right.
MC_GRP_CONTEXT_FG_PIXEL_IDX	Specifies the foreground pixel value
MC_GRP_CONTEXT_BG_PIXEL_IDX	Specifies the background pixel value
MC_GRP_CONTEXT_ALPHA_IDX	This function specifies the degree of transparency. The values used for transparency are from 0 to 255. A value of 0 means that the content is not displayed on the screen; when the value is 255, however, the content is displayed non-transparently. When specifying the degree of transparency, MC_GRP_CONTEXT_PXELOP_IDX and MC_GRP_CONTEXT_PIXEL_PARAM1_IDX are specified internally.
MC_GRP_CONTEXT_FONT_IDX	Specifies the font ID
MC_GRP_CONTEXT_STYLE_IDX	Specifies the stroke style; the value is either MC_GRP_SOLID_STYLE or MC_GRP_DOTTED_STYLE
MC_GRP_CONTEXT_PXELOP_IDX	Specifies the pixel operation function, i.e., pointer of the function
MC_GRP_CONTEXT_OFFSET_IDX	This function specifies the offset of the relative coordinates. An array of integers is placed in pv. The x coordinate of the offset is specified in the first position, and the y coordinate, in the second position.

Parameters

[in] index Index; MC_GRP_CONTEXT_CLIP_IDX,
 MC_GRP_CONTEXT_FG_PIXEL_IDX,
 MC_GRP_CONTEXT_BG_PIXEL_IDX,
 MC_GRP_CONTEXT_ALPHA_IDX,
 MC_GRP_CONTEXT_FONT_IDX,
 MC_GRP_CONTEXT_STYLE_IDX,

MC_GRP_CONTEXT_PIXELOP_IDX,
MC_GRP_CONTEXT_OFFSET_IDX

[in] pv Value corresponding to the index

Side Effects

None

Reference Item

MC_GrpContext, MC_GrpGetContext

● MC_grpGetContext

Prototype

void MC_grpGetContext(MC_GrpContext* pgc, M_Int32 index, void* pv)

Description

This function gets the value of the graphic context. The value of a specific part of the graphic context is read. The content to be placed in pv varies depending on the index value (for more information, refer to MC_grpSetContext).

Parameters

[in]	index	Index; MC_GRP_CONTEXT_CLIP_IDX, MC_GRP_CONTEXT_FG_PIXEL_IDX, MC_GRP_CONTEXT_BG_PIXEL_IDX, MC_GRP_CONTEXT_ALPHA_IDX, MC_GRP_CONTEXT_FONT_IDX, MC_GRP_CONTEXT_STYLE_IDX, MC_GRP_CONTEXT_PIXEL_OP_IDX, MC_GRP_CONTEXT_OFFSET_IDX
[out]	pv	Value corresponding to the index

Side Effects

None

Reference Item

MC_GrpContext, MC_grpSetContext

- **MC_grpPutPixel**

Prototype

```
void MC_grpPutPixel(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y,  
MC_GrpContext* pgc)
```

Description

This function puts a pixel in the specified frame buffer.

A pixel is placed in a position specified by x and y in color and transparency specified by the graphic context, which is in turn specified by pgc. When the position where the pixel is placed goes over a display, nothing is drawn in the frame buffer.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the pixel
[in]	y	Y coordinate of the pixel
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpDrawLine**

Prototype

```
void MC_grpDrawLine(MC_GrpFrameBuffer dst, M_Int32 x1, M_Int32 y1,  
M_Int32 x2, M_Int32 y2, MC_GrpContext* pgc)
```

Description

This function draws a line in a specified frame buffer.

A line connecting (x1, y2) and (x2, y2) is drawn in color, style, and transparency specified by the graphic context, which is in turn specified by pgc. When the area where the line is drawn goes over the display, nothing is drawn in the frame buffer.

Parameters

[in]	dst	Frame buffer
[in]	x1	X coordinate of the starting point
[in]	y1	Y coordinate of the starting point
[in]	x2	X coordinate of the end point
[in]	y2	Y coordinate of the end point
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpDrawRect**

Prototype

```
void MC_grpDrawRect(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y,  
M_Int32 w, M_Int32 h, MC_GrpContext* pgc)
```

Description

This function draws a rectangle in a specified frame buffer.

A rectangle with width w and height h starting from (x, y) is drawn in color, style, and transparency/drawing mode specified by the graphic context, which is in turn specified by pgc. In this case, up to (x+w-1, y+h-1) is included. When w or h is a negative number, nothing is drawn on the display.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the starting point
[in]	y	Y coordinate of the starting point
[in]	w	Width of the rectangle
[in]	h	Height of the rectangle
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpFillRect**

Prototype

```
void MC_grpFillRect(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y,  
M_Int32 w, M_Int32 h, MC_GrpContext* pgc)
```

Description

This function paints the rectangle in a specified frame buffer.

A rectangle with width *w* and height *h* starting from (*x*, *y*) is drawn in color and transparency/drawing mode specified by the graphic context, which is in turn specified by *pgc*. In this case, up to (*x*+*w*-1, *y*+*h*-1) is included, but not (*x*+*w*, *y*+*h*).

When *w* or *h* is a negative number, nothing is drawn on the display.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the starting point
[in]	y	Y coordinate of the starting point
[in]	w	Width of the rectangle
[in]	h	Height of the rectangle
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

● MC_grpCopyFrameBuffer

Prototype

```
void MC_grpCopyFrameBuffer(MC_GrpFrameBuffer dst, M_Int32 dx,
M_Int32 dy, M_Int32 w, M_Int32 h, MC_GrpFrameBuffer src, M_Int32 sx,
M_Int32 sy, MC_GrpContext* pgc)
```

Description

This function draws an image in a specified frame buffer.

The content of the (sx, sy) position is copied from the frame buffer indicated by img in an area with width w and height h starting from (dx, dy) in transparency and transparent color specified by the graphic context, which is in turn specified by pgc.

In this case, dst and src should not indicate the same buffer. Likewise, the MC_grpCopyArea function should be used.

When h is a negative number, or the area to be copied is out of the src area, nothing is drawn on the display.

Parameters

[in]	dst	Frame buffer
[in]	dx	X coordinate of the area
[in]	dy	Y coordinate of the area
[in]	w	Width of the area
[in]	h	Height of the area
[in]	src	Area of the source frame buffer
[in]	sx	X coordinate of the source area
[in]	sy	Y coordinate of the source area
[in]	pgc	Graphic context

Side Effects

None

Reference Item

MC_grpCopyArea

● MC_grpDrawImage

Prototype

```
void MC_grpDrawImage(MC_GrpFrameBuffer dst, M_Int32 dx, M_Int32 dy, M_Int32 w, M_Int32 h, MC_GrpImage src, M_Int32 sx, M_Int32 sy, MC_GrpContext* pgc)
```

Description

This function draws an image in a specified frame buffer. The content of the (sx, sy) position is copied from the frame buffer indicated by img in an area with width w and height h starting from (dx, dy) in transparency and transparent color specified by the graphic context, which is in turn specified by pgc.

In this case, if there is a mask image within the image, the content printed by this mask image is changed.

When w or h is a negative number, the area to be copied is out of the src area or mask plane, the mask is not 1-bit plane, or src does not have the same depth as the display, nothing is drawn on the display.

Parameters

[in]	ds	Frame buffer
[in]	dx	X coordinate of the area
[in]	dy	Y coordinate of the area
[in]	w	Width of the area
[in]	h	Height of the area
[in]	src	Area of the source frame buffer
[in]	sx	X coordinate of the source area
[in]	sy	Y coordinate of the source area
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

● MC_grpDrawImageRegion

Prototype

```
void MC_grpDrawImageRegion(MC_GrpFrameBuffer destbuf, M_Int32  
destX, M_Int32 destY, M_Int32 width, M_Int32 height, MC_GrpImage  
srcbuf, M_Int32 srcX, M_Int32 srcY, M_Int32 transform, MC_GrpContext  
*gc)
```

Description

This function implements a specified transformation in a specified frame buffer and draws an image.

The content of the (srcX, srcY) position is copied from the frame buffer indicated by img in a region with width w and height h starting from (destX, destY) in transparency and transparent color specified by the graphic context, which is in turn specified by gc. In this case, if there is a mask image within the image, the content printed by such mask image is changed. After a specified transformation, the content is finally copied to destbuf. Possible transformations are determined by the following transform values:

MC_GRP_TRAN_ROT90	Rotates 90° to the right
MC_GRP_TRAN_ROT180	Rotates 180° to the right
MC_GRP_TRAN_ROT270	Rotates 270° to the right
MC_GRP_TRAN_MIR	Left-right mirroring
MC_GRP_TRAN_MIR_ROT90	Rotates 90° to the right after left-right mirroring
MC_GRP_TRAN_MIR_ROT180	Rotates 180° to the right after left-right mirroring
MC_GRP_TRAN_MIR_ROT270	Rotates 270° to the right after left-right mirroring

When w or h is a negative number, the area to be copied is out of the src area (region??) or mask plane, the mask is not 1-bit plane, or src does not have the same depth as the display, nothing is drawn on the display.

Parameters

[in]	destbuf	Frame buffer
[in]	destX	X coordinate of the area
[in]	destY	Y coordinate of the area
[in]	width	Width of the area
[in]	height	Height of the area
[in]	srcbuf	Region of the source frame buffer
[in]	srcX	X coordinate of the source region

[in]	srcY	Y coordinate of the source region
[in]	transform	Specifications for rotation, mirror effect, etc.
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpCopyArea**

Prototype

```
void MC_grpCopyArea(MC_GrpFrameBuffer dst, M_Int32 dx, M_Int32 dy,  
M_Int32 w, M_Int32 h, M_Int32 x, M_Int32 y, MC_GrpContext* pgc)
```

Description

This function copies the content of a specified frame to itself.

The content of width w and height h is copied from (x, y) of dst to (dx, dy) of the frame buffer indicated by dst.

Parameters

[in]	dst	Frame buffer
[in]	dx	X coordinate of the image to be copied
[in]	dy	Y coordinate of the image to be copied
[in]	w	Width of the image to be copied
[in]	h	Height of the image to be copied
[in]	x	X coordinate of the area to be copied
[in]	y	Y coordinate of the area to be copied
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpDrawArc**

Prototype

```
void MC_grpDrawArc(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y,  
M_Int32 w, M_Int32 h, M_Int32 s, M_Int32 e, MC_GrpContext* pgc)
```

Description

This function draws an arc in a specified frame buffer.

An arc is drawn in style, transparency, and color/drawing mode specified by the graphic context, which is in turn specified by pgc.

In this case, the arc is drawn from s up to e, with 0° angle pointing to the 3 o'clock direction. A positive value indicates a counterclockwise direction, whereas a negative value indicates a clockwise direction.

The center of a rectangle with starting point (x, y), height h, and width w becomes the center of the arc.

Similar to the case in drawRect, the arc occupies an area whose height is (x + width, y + height). When the width or height is smaller than 0, nothing is drawn.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the area
[in]	y	Y coordinate of the area
[in]	w	Width of the arc
[in]	h	Height of the arc
[in]	s	Starting angle (0-360°)
[in]	e	Arc angle (0-360°)
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

● MC_grpFillArc

Prototype

```
void MC_grpFillArc(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y,  
M_Int32 w, M_Int32 h, M_Int32 s, M_Int32 e, MC_GrpContext* pgc)
```

Description

This function paints an arc in a specified frame buffer.

An arc is painted in transparency and color/drawing mode specified by the graphic context, which is in turn specified by pgc.

In this case, the arc is drawn from s up to e, with 0° angle pointing to the 3 o'clock direction. A positive value indicates a counterclockwise direction, whereas a negative value indicates a clockwise direction.

The center of a rectangle with starting point (x, y), height h, and width w becomes the center of the arc.

Similar to the case in fillRect, the arc occupies an area whose height is (x + width - 1, y + height - 1). When the width or height is smaller than 0, nothing is drawn.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the area
[in]	y	Y coordinate of the area
[in]	w	Width of the area
[in]	h	Height of the area
[in]	s	Starting angle (0-360°)
[in]	e	Arc angle (0-360°)
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpDrawString**

Prototype

```
void MC_grpDrawString(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y,  
const M_Char* str, M_Int32 len, MC_GrpContext* pgc)
```

Description

This function draws a string in a specified frame buffer.

A string is drawn in transparency and color/drawing mode and with a font specified by the graphic context, which is in turn specified by pgc.

str is a common C string. To print a Unicode string, the drawUnicodeString function is used.

When len is -1, the string is processed until the character becomes null. On the other hand, when len is 0 or a negative number other than -1, nothing is printed on the display.

Parameters

[in]	pd	Frame buffer
[in]	bpl	Number of bytes per line of the frame buffer
[in]	x	X coordinate of the string
[in]	y	Y coordinate of the baseline of the string
[in]	str	C string
[in]	len	Length of the string
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpDrawUnicodeString**

Prototype

```
void MC_grpDrawUnicodeString(MC_GrpFrameBuffer dst, M_Int32 x,  
M_Int32 y, const M_UCode* str, M_Int32 len, MC_GrpContext* pgc)
```

Description

This function draws a string in a specified frame buffer.

A string is drawn in transparency and color/drawing mode and with a font specified by the graphic context, which is in turn specified by pgc.

str is a Unicode string. To print a C string, the drawString function is used. In an ordinary program, this function is not necessary. When writing a Java program, especially when writing a program using a combination of Java and C language, this function is provided to expedite the drawing.

When len is a negative number, nothing is printed on the display.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the string
[in]	y	Y coordinate of the baseline of the string
[in]	str	Unicode string
[in]	len	Length of the string
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpGetRGBPixels**

Prototype

void MC_grpGetRGBPixels(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y, M_Int32 w, M_Int32 h, M_Uint32* pd, M_Int32 ipl)

Description

This function brings various pixel colors of a specified frame buffer at the same time.

Various color values of any frame buffer area are copied to an integer array. The color value is in the form 0x00RRGGBB.

When the area brought is out of bounds of the area, or its width is bigger than ipl, the value to be copied is determined depending on every program implementation.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the area to be brought
[in]	y	Y coordinate of the area to be brought
[in]	w	Width of the area to be brought
[in]	h	Height of the area to be brought
[out]	pd	Integer array
[in]	ipl	Number of integers per line of the image

Side Effects

None

Reference Item

None

● MC_grpSetRGBPixels

Prototype

```
void MC_grpSetRGBPixels(MC_GrpFrameBuffer dst, M_Int32 x, M_Int32 y, M_Int32 w, M_Int32 h, const M_Uint32* psrc, M_Int32 ibpl, MC_GrpContext* pgc)
```

Description

This function specifies various pixel colors of a specified frame buffer at the same time.

Various color values stored in an integer array are copied to any frame buffer area. Depending on the number of colors in LCD, the colors are changed to the nearest colors before they are set. The color value is in the form 0x00RRGGBB.

When the area to be brought is out of bounds of the frame buffer, or ibpl has a value that is smaller than w, the image is not copied properly.

Parameters

[in]	dst	Frame buffer
[in]	x	X coordinate of the area to be modified
[in]	y	Y coordinate of the area to be modified
[in]	w	Width of the area to be modified
[in]	h	Height of the area to be modified
[in]	pd	Integer array
[in]	ibpl	Number of bytes per line of the image
[in]	pgc	Graphic context

Side Effects

None

Reference Item

None

- **MC_grpFlushLcd**

Prototype

```
void MC_grpFlushLcd(M_Int32 i, MC_GrpFrameBuffer frm, M_Int32 x,  
M_Int32 y, M_Int32 w, M_Int32 h)
```

Description

This function prints a specified frame buffer to the LCD display.

Part of the content of any frame buffer is printed to the LCD display. Here, the number of bytes per line of the frame buffer and the number of bytes per line of the LCD should be identical.

The user can use several display frame buffers or just one display frame buffer.

Parameters

[in]	pd	Frame buffer
[in]	i	Display index (main display when the value is 0; auxiliary LCD display when the value is 1)
[in]	bpl	Number of bytes per line of the frame buffer
[in]	x	X coordinate of the area to be printed
[in]	y	Y coordinate of the area to be printed
[in]	w	Width of the area to be printed
[in]	h	Height of the area to be printed

Side Effects

None

Reference Item

None

- **MC_grpGetPixelFromRGB**

Prototype

M_Int32 MC_grpGetPixelFromRGB(M_Int32 r, M_Int32 g, M_Int32 b)

Description

Gets the pixel value for a specified color; when r, g, or b exceeds 255, an appropriate pixel value cannot be obtained

Parameters

[in]	r	Red value (0-255)
[in]	g	Green value (0-255)
[in]	b	Blue value (0-255)

Return Value

Pixel value

Side Effects

None

Reference Item

None

- **MC_grpGetRGBFromPixel**

Prototype

*M_Int32 MC_grpGetRGBFromPixel(M_Int32 pixel, M_Int32 *r, M_Int32 *g, M_Int32 *b)*

Description

Gets color values for a specified pixel value

Parameters

[in]	pixel	Pixel value
[out]	r	Red value (0-255)
[out]	g	Green value (0-255)
[out]	b	Blue value (0-255)

Return Value

Same as the pixel of the pixel value

Side Effects

None

Reference Item

None

- **MC_grpGetDisplayInfo**

Prototype

M_Int32 MC_grpGetDisplayInfo(M_Int32 i, MC_GrpDisplayInfo pi)*

Description

Gets a display information structure

Parameters

[in]	i	Reserved parameter; 0 is put
[out]	pi	Display information structure

Return Value

Returns 1 in case of a corresponding information structure; otherwise, 0 is returned

Side Effects

None

Reference Item

None

- **MC_grpRepaint**

Prototype

```
void MC_grpRepaint(M_Int32 lcd, M_Int32 x, M_Int32 y, M_Int32 w,  
M_Int32 h)
```

Description

This function causes the paintClet function to be called for a specific area. When this function is called, paintClet is not called directly within the function; instead, a related event is put in a program event queue to enable paintClet to be called.

Parameters

[in]	l	Main LCD display frame buffer when the display frame buffer is 0; auxiliary LCD display frame buffer when the display frame buffer is 1
[in]	x	X coordinate of the area
[in]	y	Y coordinate of the area
[in]	w	Width of the area
[in]	h	Height of the area

Side Effects

None

Reference Item

None

- **MC_grpGetFont**

Prototype

M_Int32 MC_grpGetFont(M_Int32 face, M_Int32 size, M_Int32 style)

Description

Returns the closest system font to the specified font; when an appropriate face, size, or style is not received, the default font is returned

Parameters

[in]	face	Font face; MC_GRP_FT_FACE_SYSTEM, MC_GRP_FT_FACE_MONOSPACE, or MC_GRP_FT_FACE_PROPORTIONAL
[in]	size	Font size; MC_GRP_FT_SIZE_LARGE, MC_GRP_FT_SIZE_MEDIUM, or MC_GRP_FT_SIZE_SMALL
[in]	style	Font style; MC_GRP_FT_STYLE_PLAIN, MC_GRP_FT_STYLE_ITALIC, MC_GRP_FT_STYLE_BOLD, or MC_GRP_FT_STYLE_UNDERLINED

Return Value

Specified font ID

Side Effects

None

Reference Item

None

- **MC_grpGetFontHeight**

Prototype

M_Int32 MC_grpGetFontHeight(M_Int32 font)

Description

Returns the font height; when the specified font is not valid, 0 is returned

Parameters

[in] font Font ID

Return Value

Font height

Side Effects

None

Reference Item

None

- **MC_grpGetFontAscent**

Prototype

M_Int32 MC_grpGetFontAscent(M_Int32 font)

Description

Returns the font ascent; when the specified font is not valid, 0 is returned

Parameters

[in] font Font ID

Return Value

Font ascent

Side Effects

None

Reference Item

None

- **MC_grpGetFontDescent**

Prototype

M_Int32 MC_grpGetFontDescent(M_Int32 font)

Description

Returns the font descent; when the specified font is not valid, 0 is returned

Parameters

[in] font Font ID

Return Value

Success

Font descent

Failure

M_E_INVALID

Issued in case the specified font is invalid

Side Effects

None

Reference Item

None

- **MC_grpGetStringWidth**

Prototype

M_Int32 MC_grpGetStringWidth(M_Int32 font, const M_Uint8 str, M_Int32 len)*

Description

This function gets the width of a string on the display.

The width of a string is returned on the display when the string is printed on the display using a specified font.

A len value of -1 returns the width up to the end of a string (where null is displayed).

Parameters

[in]	font	Font ID
[in]	str	String
[in]	len	Length of the string

Return Value

Pass	Width of the string
Fail	M_E_INVALID wrong parameter

Side Effects

None

Reference Item

None

- **MC_grpGetUnicodeStringWidth**

Prototype

M_Int32 MC_grpGetUnicodeStringWidth(M_Int32 font, const M_UCode* str, M_Int32 len)

Description

This function gets the width of a Unicode string on the display.

The width of a Unicode string is returned on the display when the string is printed on the display using a specified font.

A len value of -1 returns the width up to the end of a string (where null is displayed).

Parameters

[in]	font	Font
[in]	str	String
[in]	len	Length of a string

Return Value

Pass	Width of the string
Fail	M_E_INVALID wrong parameter

Side Effects

None

Reference Item

None

● MC_grpCreateImage

Prototype

M_Int32 MC_grpCreateImage(MC_GrpImage *newImg, M_Uint32 bufID, M_Int32 off, M_Int32 len)

Description

This function creates an image.

An image structure is initialized by passing a buffer with image data (PNG, BMP format).

When calling this function, an image is created first. When the image is not created properly, null is returned.

The buf passed as a parameter is automatically released within the image function. Therefore, the buf should have the ID of a buffer that was created using MC_knlCalloc. For the time of release, the buf is released after the initial frame is completed for a non-animation image, and when the MC_grpDestroyImage function is called for an animation image.

Parameters

[out]	newImg	New image created; in case of error, it is set as null
[in]	bufID	Buffer with image data; it should have the ID of a buffer that was allocated using MC_knlCalloc
[in]	off	Offset of the image data
[in]	len	Length of the image data

Return Value

MC_GRP_IMAGE_DONE	Issued when the image is decoded without an error
M_E_NOMEMORY	Issued in case of insufficient memory for creating an image
M_E_BADFORMAT	Issued when the image format is invalid

Side Effects

None

Reference Item

None

● MC_grpCreateSubImage

Prototype

***MC_GrpImage MC_grpCreateSubImage(MC_GrpImage srcImage,
M_Int32 startx, M_Int32 starty, M_Int32 width, M_Int32 height)***

Description

This function creates a sub-image. When the image is not properly created, null is returned. When a sub-image with width w and height h is created from (startx, starty) of a source image, and if there is a mask in the source image, a mask will also be created in the sub-image.

Parameters

[in]	srcImage	Original image structure from which a sub-image is created
[in]	startx, starty	Reference coordinates of the original image; this serves as (0, 0) coordinates of the sub-image
[in]	width, height	Width and height of the sub-image to be created

Return Value

Sub-image structure

Side Effects

None

Reference Item

None

- **MC_grpDestroyImage**

Prototype

void MC_grpDestroyImage(MC_GrpImage img)

Description

Removes an image

Parameters

img Image to be removed

Side Effects

None

Reference Item

None

● MC_grpDecodeNextImage

Prototype

M_Int32 MC_grpDecodeNextImage(MC_GrpImage dst)

Description

This function creates the next frame of an image.

Data is read from a specific buffer to create an image. The image created goes into an image target structure.

In case of a moving image, the next image is decoded and stored in dst when this function is called once more. Note that the dst where the first image is stored should be passed, because there are cases when the next image is created by drawing only the changed image over the previous image, considering the characteristics of a moving image.

Parameters

[out] dst Image target structure

Return Value

The return value will be any one of the following four:

MC_GRP_FRAME_DONE	Issued when the image is decoded without an error
MC_GRP_IMAGE_DONE	Issued when the image is decoded without an error, and there is no more image to be decoded
M_E_NOMEMORY	Issued in case of insufficient memory for creating an image
M_E_BADFORMAT	Issued when the image format is invalid

Side Effects

None

Reference Item

None

- **MC_grpEncodeImage**

Prototype

M_Int16 MC_grpEncodeImage(MC_GrpFrameBuffer src, M_Int32 x, M_Int32 y, M_Int32 w, M_Int32 h, M_Int32* len)

Description

This function encodes a certain area of a specified frame buffer in graphic file format.

A certain area of the specified frame buffer is encoded in BMP file format and returned.

The size of the returned image buffer is obtained as a len parameter. The value returned is a buffer ID created through MC_knlCalloc.

Parameters

[in]	src	Image source structure
[in]	x	Starting x coordinate of the area
[in]	y	Starting y coordinate of the area
[in]	w	Width of the area
[in]	h	Height of the area
[out]	len	Size of the image buffer encoded

Return Value

ID of the image buffer encoded; in case of failure, null is returned

Side Effects

None

Reference Item

None

- **MC_grpPostEvent**

Prototype

M_Int32 MC_grpPostEvent(M_Int32 id, M_Int32 type, M_Int32 param1, M_Int32 param2)

Description

This function puts an event in the event queue of a specified program.

An event is put in the event queue to enable the specified program to receive it. In the specified program, the relevant event occurs, and the handleCletEvent function is called. Each parameter becomes type, param1, and param2.

Parameters

[in]	id	Program ID
[in]	type	Event type
[in]	param1	Parameter 1
[in]	param2	Parameter 2

Return Value

Pass	1
Fail	0

Side Effects

None

Reference Item

None

- **MC_grpDrawPolygon**

Prototype

```
void MC_grpDrawPolygon(MC_GrpFrameBuffer dst, M_INT32* xPoints,  
M_INT32* yPoints, M_Int32 nPoints, MC_GrpContext* pgc)
```

Description

Draws a polygon with random vertexes

Parameters

[in]	dst	Frame buffer
[in]	xPoints	Array of x coordinates
[in]	yPoints	Array of y coordinates
[in]	nPoints	Number of vertices
[in]	pgc	Graphic context

Return Value

None

Side Effects

None

Reference Item

MC_grpDrawFillPolygon

- **MC_grpFillPolygon**

Prototype

```
void MC_grpDrawFillPolygon(MC_GrpFrameBuffer dst, M_INT32*  
xPoints, M_INT32* yPoints, M_INT32 nPoints, MC_GrpContext* pgc)
```

Description

Draws a filled polygon with random vertices

Parameters

[in]	dst	Frame buffer
[in]	xPoints	Array of x coordinates
[in]	yPoints	Array of y coordinates
[in]	nPoints	Number of vertices
[in]	pgc	Graphic context

Return Value

None

Side Effects

None

Reference Item

MC_grpDrawPolygon

- **MC_imGetSupportModeCount**

Prototype

M_Int32 MC_imGetSupportModeCount()

Description

Gets the number of input modes supported by the automata

Parameters

None

Return Value

Number of input modes

Side Effects

None

Reference Item

None

- **MC_imGetSupportedModes**

Prototype

M_Char** MC_imGetSupportedModes()

Description

This function gets the language code of input modes supported by the automata. The language code should be in compliance with the ISO 639 Code. In case the language differentiates the upper case and lower case characters, however, "/S" and "/L" can be added to each language code. For example, in case of English lower case characters, a language code of "EN/S" is passed, and in case of Korean characters, a language code of "KO" is passed.

Parameters

None

Return Value

Language code (string array pointer)

Side Effects

None

Reference Item

None

- **MC_imSetCurrentMode**

Prototype

M_Int32 MC_imSetCurrentMode (M_Int32 mode)

Description

This function sets the mode to be used in the automata. This value is the index value of a language code obtained through MH_IMAgetSupportedModes().

Parameters

[in] mode Input mode

Return Value

Returns 1 when the set input mode is properly applied; otherwise, 0 is returned

Side Effects

None

Reference Item

None

- **MC_imGetCurrentMode()**

Prototype

M_Int32 MC_imGetCurrentMode()

Description

This function gets the current input mode of the automata. This value is the index value of a language code obtained through MH_IMAgetSupportedModes().

Parameters

None

Return Value

Current input mode of the automata

Reference Item

None

● MC_imHandleInput

Prototype

M_Int32 MC_imHandleInput(M_Char key, M_Int32 type, char *buf1, M_Int32 *size1, char *buf2, M_Int32 *size2)

Description

This function handles key inputs received from a user component according to the current input mode, creates characters, and passes the created characters. In case the transmitted key is one that cannot be used for a combination of the automata, this function sets the character combined in a completed buffer and returns 0.

Parameters

[in]	key	Key value inputted (defined in MH_KeyCode and MH_IMA_FLUSH)
[in]	type	Key type inputted (defined in MH_Event)
[out]	buf1	Completed string buffer
[in]	size1	Size of the completed string buffer
[out]	buf2	String buffer combined
[in]	size2	Size of the string buffer combined

Return Value

Pass		
	1	When the automata handled the input
Fail		
	0	When the automata cannot handle the input

Reference Item

None

2.3. File

This is a module related to the files.

Here, the operating system is assumed to support a file system with a hierarchical directory structure.

This module includes APIs that are related to opening, reading, and writing files as well as reading file information.

All filenames consist of absolute paths. In actuality, all filenames can be created and removed in a directory allowed by the platform. The user only has to use the absolute paths regardless of the way filenames are supported by the system.

For the separator, the present protocol follows the practice employed by the Unix system, which allows the use of "/."

For the following APIs that use the file path as an argument, access level should also be put as an argument:

MC_fsOpen

MC_fsFileAttribute

MC_fsRemove

MC_fsRename

MC_fsMkDir

MC_fsRmdir

MC_fsList

MC_fsSetMode

MC_fsGetCounts

MC_fsIsExist

There are three levels of access:

MC_DIR_PRIVATE_ACCESS: Access to a private directory

MC_DIR_SHARED_ACCESS: Access to a shared directory

MC_DIR_SYSTEM_ACCESS: Access to a system directory

All programs can access their own private directories. For other levels of access, only authorized programs can access relevant directories. In case a file is shared, when a program is open for Read/Write, other programs can access it as Read-Only. All the abovementioned APIs are blocking APIs. In other words, when all file operations are completed, they are returned.

The following is the structure of MC_FileInfo:

```
struct _fileInfo{
    M_Int32    attrib;           // Bit masks that display the
file attributes
    M_Uint32   creationTime; // The time when the file was
created is displayed in seconds since Jan. 1, 1970.
    M_Uint32   size;           // File size
};
```

If the MC_FILE_IS_DIR bit is set in fileInfo->attrib, it is a directory.

- **MC_FILE_OPEN_RDONLY**

Prototype

#define MC_FILE_OPEN_RDONLY

Description

Only reading is possible. This is defined as MH_FILE_OPEN_RDONLY.

- **MC_FILE_OPEN_WRONLY**

Prototype

#define MC_FILE_OPEN_WRONLY

Description

Only writing is possible. This is defined as MH_FILE_OPEN_WRONLY.

- **MC_FILE_OPEN_WRTRUNC**

Prototype

#define MC_FILE_OPEN_WRTRUNC

Description

Only opening is possible. When a file exists, its size is rendered to 0. This is defined as MH_FILE_OPEN_WRTRUNC.

- **MC_FILE_OPEN_RDWR**

Prototype

#define MC_FILE_OPEN_RDWR

Description

Both reading and writing are possible. This is defined as MH_FILE_OPEN_RDWR.

- **MC_FILE_IS_DIR**

Prototype

#define MC_FILE_IS_DIR

Description

This is a bit that represents a directory among the file attributes. This is defined as MH_FILE_IS_DIR.

- **MC_MAX_FILENAME_LENGTH**

Prototype

#define MC_MAX_FILENAME_LENGTH

Description

Maximum length of a filename; this is defined as MH_MAX_FILENAME_LENGTH

- **MC_DIR_PRIVATE_ACCESS**

Prototype

#define MC_DIR_PRIVATE_ACCESS

Description

Access to one's own directory; this is defined as 1

- **MC_DIR_SHARED_ACCESS**

Prototype

#define MC_DIR_SHARED_ACCESS

Description

Access to a shared directory; this is defined as 2

- **MC_DIR_SYSTEM_ACCESS**

Prototype

#define MC_DIR_SYSTEM_ACCESS

Description

Access to a system directory; this is defined as 3

- **MC_FILE_SEEK_SET**

Prototype

#define MC_FILE_SEEK_SET

Description

The position of the file pointer is set based on the start of a file. This is defined as MH_FILE_SEEK_SET.

- **MC_FILE_SEEK_CUR**

Prototype

#define MC_FILE_SEEK_CUR

Description

The position of the file pointer is set based on the current position of a file. This is defined as MH_FILE_SEEK_CUR.

- **MC_FILE_SEEK_END**

Prototype***#define MC_FILE_SEEK_END*****Description**

The position of the file pointer is set based on the end of a file. This is defined as MH_FILE_SEEK_END.

● MC_fsOpen

Prototype

M_Int32 MC_fsOpen(M_Char* name, M_Int32 mode, M_Int32 accessLevel)

Description

This function opens a file. The position of a directory to be opened varies depending on the value of accessLevel. A filename is limited to 30 characters.

Parameters

name	Filename	
mode	MC_FILE_OPEN_RDONLY	Only reading is possible.
	MC_FILE_OPEN_WRONLY	Only writing is possible.
	MC_FILE_OPEN_WRTRUNC	Only opening is possible. When a file exists, its size is rendered to 0.
	MC_FILE_OPEN_RDWR	Both reading and writing are possible.
	accessLevel	
	MC_DIR_PRIVATE_ACCESS	Access to a private directory
	MC_DIR_SHARED_ACCESS	Access to shared directory
	MC_DIR_SYSTEM_ACCESS	Access to system directory

Return Value

Pass	File ID	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_NOENT	Issued in the absence of a file when trying to open a file through MC_FILE_OPEN_RDONLY
	M_E_BADFILENAME	Issued when the filename is invalid
	M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
	M_E_INVALID	Issued when the mode is invalid
	M_E_NOSPACE	Issued in the absence of available space in the file system
	M_E_ACCESS	Issued when a file cannot be accessed

Side Effects

None

Reference Item

Refer to the meaning of mkdir accessLevel.

● MC_fsRead**Prototype**

M_Int32 MC_fsRead(M_Int32 fd, M_Byte* buf, M_Int32 len)

Description

Reads a file; reads as much byte as the len size with buf

Parameters

fd	File ID
buf	Buffer where the data read from a file are stored
len	Number of bytes to be read

Return Value

Pass	Number of bytes actually read	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_BADFD	Issued when the File ID is invalid
	M_E_EOF	Issued when end of file (EOF) is reached

Side Effects

None

Reference Item

None

● MC_fsWrite

Prototype

M_Int32 MC_fsWrite(M_Int32 fd, M_Byte* buf, M_Int32 len)

Description

Writes in a file; writes as much byte of the buf content as the len size

Parameters

fd	File ID
buf	Buffer where the data to be written in a file are stored
len	Number of bytes to be written

Return Value

Pass	Number of bytes actually written	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_BADFD	Issued when the File ID is invalid
	M_E_NOSPACE	Issued in the absence of available space in the file system

Side Effects

None

Reference Item

None

- **MC_fsClose**

Prototype

M_Int32 MC_fsClose(M_Int32 fd)

Description

Closes a file

Parameters

fd File ID

Return Value

Pass

0

Fail

M_E_ERROR

Issued in case of failure due to other reasons

M_E_BADFD

Issued when the File ID is invalid

Side Effects

None

Reference Item

None

● MC_fsSeek

Prototype

M_Int32 MC_fsSeek(M_Int32 fd, M_Int32 pos, M_Int32 where)

Description

Adjusts the position of the file pointer indicating the current position of reading or writing

Parameters

fd	File ID	
pos	Position of the file pointer to be set	
where	MC_FILE_SEEK_SET	Position of the file pointer that is set based on the start of a file
	MC_FILE_SEEK_CUR	Position of the file pointer that is set based on the current position of a file
	MC_FILE_SEEK_END	Position of the file pointer that is set based on the end of a file

Return Value

Pass	Current position of the file pointer	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_BADFD	Issued when the File ID is invalid
	M_E_INVALID	Issued when where invalid
	M_E_BADSEEKPOS	Issued when the file pointer is out of bounds for the file size

Side Effects

None

Reference Item

None

● **MC_fsFileAttribute**

Prototype

M_Int32 MC_fsFileAttribute(M_Char fileName, MC_FileInfo* fileInfo, M_Int32 accessLevel)*

Description

This function gets the current file attributes.

The bit mask that indicates the file attributes in the MC_FileInfo structure is used as follows:

```
program A

MC_FileInfo fi;
...
int rtn = MC_fsFileAttribute("B", &fi, MC_DIR_PRIVATE_ACCESS);
if (fi.attrib & MC_FILE_IS_DIR) {
    // This refers to a directory.
} else {
}
...

```

Parameters

fileName	Filename	
fileInfo	Pointer of the file attribute structure	
accessLevel	MC_DIR_PRIVATE_ACCESS	Access to a private directory
	MC_DIR_SHARED_ACCESS	Access to shared directory
	MC_DIR_SYSTEM_ACCESS	Access to system directory

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_BADFILENAME	Issued when the filename is invalid
	M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
	M_E_ACCESS	Issued when a file cannot be accessed

Side Effects

None

Reference Item

Refer to the meaning of mkdir accessLevel.

● MC_fsRemove

Prototype

M_Int32 MC_fsRemove(M_Char* fileName, M_Int32 accessLevel)

Description

This function removes a file. A file that is open cannot be removed, however.

Parameters

fileName	Filename
accessLevel	MC_DIR_PRIVATE_ACCESS Access to a private directory
	MC_DIR_SHARED_ACCESS Access to shared directory
	MC_DIR_SYSTEM_ACCESS Access to system directory

Return Value

Pass	0
Fail	
	M_E_ERROR Issued in case of failure due to other reasons
	M_E_BADFILENAME Issued when the filename is invalid
	M_E_LONGNAME Issued when the length of a filename exceeds the maximum length
	M_E_NOENT Issued when the file does not exist.
	M_E_ACCESS Issued when a file cannot be accessed

Side Effects

None

Reference Item

Refer to the meaning of mkdir accessLevel.

● MC_fsRename

Prototype

M_Int32 MC_fsRename(M_Char* oldname, M_Char* newname, M_Int32 accessLevel)

Description

Changes a filename; when the new filename to be used already exists, the filename cannot be changed

Parameters

oldname	Old filename
newname	New filename
accessLevel	MC_DIR_PRIVATE_ACCESS Access to a private directory
	MC_DIR_SHARED_ACCESS Access to shared directory
	MC_DIR_SYSTEM_ACCESS Access to system directory

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_BADFILENAME	Issued when the filename is invalid
	M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
	M_E_EXIST	New filename to be used already exists
	M_E_NOSPACE	No remaining space in the file system
	M_E_ACCESS	Issued when the file cannot be accessed

Side Effects

None

Reference Item

Refer to the meaning of mkdir accessLevel.

● MC_fsMkDir

Prototype

M_Int32 MC_fsMkDir(M_Char* dirName, M_Int32 accessLevel)

Description

Creates a directory; the program can create a directory in an area with different access right requirements based on the value of accessLevel.

Parameters

dirName	Directory name
accessLevel	MC_DIR_PRIVATE_ACCESS Access to a private directory
	MC_DIR_SHARED_ACCESS Access to shared directory
	MC_DIR_SYSTEM_ACCESS Access to system directory

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_EXIST	Issued when the directory already exists
	M_E_NOENT	Issued when the parent director does not exist
	M_E_BADFILENAME	Issued when the filename is invalid
	M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
	M_E_ACCESS	Issued when the file cannot be accessed

Side Effects

None

Reference Item

None

● MC_fsRmdir

Prototype

M_Int32 MC_fsRmdir(M_Char* dirName, M_Int32 accessLevel)

Description

This function removes a directory. In case of remaining files or sub-directories in the directory, however, the directory cannot be removed.

Parameters

dirName Directory name

accessLevel	MC_DIR_PRIVATE_ACCESS	Access to a private directory
	MC_DIR_SHARED_ACCESS	Access to shared directory
	MC_DIR_SYSTEM_ACCESS	Access to system directory

Return Value

Pass

0

Fail

M_E_ERROR	Issued in case of failure due to other reasons
M_E_NOENT	Issued when the directory does not exist
M_E_NOTEMPTY	Issued when there are files and directories in the directory
M_E_BADFILENAME	Issued when the filename is invalid
M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
M_E_ACCESS	Issued when the file cannot be accessed

Side Effects

None

Reference Item

Refer to the meaning of mkdir accessLevel.

● MC_fsList

Prototype

M_Int32 MC_fsList(M_Char* fileName, M_Char* buf, M_Int32 bufSize, M_Int32 accessLevel)

Description

This function gets a list of files and directories existing in a directory. Names for files and directories are stored continuously in the buffer as a null terminator string and returned. At the end of the list, null comes twice.

For example, when there are two files in the directory named file1 and file2:

file1\0file2\0\0

Parameters

fileName	Directory name	
buf	Buffer where names for files and directories are stored and returned	
bufSize	Buffer size	
accessLevel	MC_DIR_PRIVATE_ACCESS	Access to a private directory
	MC_DIR_SHARED_ACCESS	Access to a shared directory
	MC_DIR_SYSTEM_ACCESS	Access to a system directory

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to other reasons
	M_E_BADFILENAME	Issued when the filename is invalid.
	M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
	M_E_SHORTBUF	Issued in case of insufficient buffer size
	M_E_ACCESS	Issued when the file cannot be accessed

Side Effects

None

Reference Item

Refer to the meaning of mkdir accessLevel.

- **MC_fsTotalSpace**

Prototype

M_Int32 MC_fsTotalSpace()

Description

Gets the total space of the file system

Return Value

Pass

Total space of the file system (byte)

Fail

M_E_ERROR Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

- **MC_fsAvailable**

Prototype***M_Int32 MC_fsAvailable()*****Description**

Gets the available space in the file system

Return Value

Pass

Size of the available space in the file system (byte)

Fail

M_E_ERROR

Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

● MC_fsSetMode

Prototype

M_Int32 MC_fsSetMode(char* fileName, M_Int32 fileMode, M_Int32 accessLevel)

Description

This function changes a file attribute. For attributes that can be changed, refer to the fileMode values in the parameter table shown below. Basically, all programs can access their own private directories. For other levels of access, however, only authorized programs can access relevant directories.

Parameters

[in] filename Absolute path name of the file

[in] fileMode File attribute mode

fileMode	Meaning	Value
MH_FILEMODE_RDONLY	To be set when the mode is Read-Only	0x10
MH_FILEMODE_WRONLY	To be set when the mode is Write-Only	0x20
MH_FILEMODE_RDWR	To be set when the mode is Read/Write	0x30

[in] accessLevel MC_DIR_PRIVATE_ACCESS Access to a private directory
 MC_DIR_SHARED_ACCESS Access to a shared directory
 MC_DIR_SYSTEM_ACCESS Access to a system directory

Return Value

Pass

M_E_SUCCESS The attribute was changed.

Fail

M_E_ERROR Issued in case of failure due to other reasons

M_E_BADPATHNAME Issued when the absolute path name is invalid

M_E_LONGNAME Issued when the length of a filename exceeds the maximum length

M_E_INVALID Issued when the mode is invalid

M_E_ACCESS Issued when the file cannot be accessed

M_E_NOENT Issued when the file does not exist

Side Effects

None

Reference Item

None

● MC_fsGetCounts

Prototype

M_int32 MC_fsGetCounts(char* dirName, M_Int32 accessLevel)

Description

Gets the number of files in the directory; the value for the number of files includes directories

Parameters

dirName Directory name

accessLevel	MC_DIR_PRIVATE_ACCESS	Access to a private directory
	MC_DIR_SHARED_ACCESS	Access to a shared directory
	MC_DIR_SYSTEM_ACCESS	Access to a system directory

Return Value

Pass

Number of files and directories

Fail

M_E_ERROR	Issued in case of failure due to other reasons
M_E_BADFILENAME	Issued when the filename is invalid
M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
M_E_ACCESS	Issued when the file cannot be accessed

Side Effects

None

Reference Item

None

● MC_fsTell**Prototype*****M_int32 MC_fsTell(M_int32 fd)*****Description**

Brings the current position in the file, serving as the reference for Read and Write operations

Parameters

fd File ID

Return Value

Pass

Position of the file pointer

Fail

M_E_ERROR

Issued in case of failure due to other reasons

M_E_BADFD

Issued when the File ID is invalid

Side Effects

None

Reference Item

None

● MC_fsIsExist

Prototype

M_Int32 MC_fsIsExist(char* fileName, M_Int32 accessLevel)

Description

Indicates whether or not a file exists on a specific path

Parameters

[in]	fileName	Filename	
[in]	accessLevel	MC_DIR_PRIVATE_ACCESS	Access to a private directory
		MC_DIR_SHARED_ACCESS	Access to a shared directory
		MC_DIR_SYSTEM_ACCESS	Access to a system directory

Return Value

Pass

M_E_SUCCESS The file exists.

Fail

M_E_ERROR	Issued in case of failure due to other reasons
M_E_BADFILENAME	Issued when the filename is invalid
M_E_LONGNAME	Issued when the length of a filename exceeds the maximum length
M_E_ACCESS	Issued when the file cannot be accessed
M_E_NOENT	Issued when the file does not exist

Side Effects

None

Reference Item

None

2.4. Database

APIs are used to store data by record as a unit as well as to search and manage.

Records are stored in the form of a character array. The meaning of data stored is irrelevant, since grasping the logical meaning of data stored is the responsibility of the user. Record size cannot exceed the size specified when the database was first created. Each record is represented by an integer value called Record ID in the database. When a record is deleted from the database, the Record ID deleted is re-used when the next record is stored. Record ID cannot be a negative number.

Due to frequent addition and removal, a difference may arise between the number of records stored in the database and the volume of the database that is actually stored in the physical area of the platform.

A program can create a number of databases and can access all of them. In most cases, however, it cannot access other databases that are created by other programs. A database can be shared by a number of programs by creating it in a shared directory; when the program has the authority, it can access a database used by the system program by specifying a flag when opening the database.

All database functions are returned once the implementation is completed. For example, the MC_dbInsertRecord function is returned after the data is written in the physical storage area. Therefore, as soon as this function is returned, the record that was stored immediately before can be brought through MC_dbSelectRecord.

Even when the program is terminated, a database can be preserved in the same state as when the implementation was completed and can be used in the next implementation. Once the program is removed from the platform, however, all databases that are created by the program are deleted.

● MC_dbOpenDataBase

Prototype

M_Int32 MC_dbOpenDataBase(M_Char* dataBaseName, M_Int32 recordSize, M_Boolean create, M_Int32 mode)

Description

This function opens a database.

The share mode of a database can be specified with a flag. Only records with a fixed size are supported. A record is represented by a consecutive character array, and the logical meaning of a record should be grasped by the database user.

Parameters

dataBaseName	Database name; characters that can be used in a database name are subject to the restrictions placed on a file system
recordSize	Size of one record (byte) in a database to be created; when a database already exists, the specified recordSize is ignored, and the existing record size is applied
create	Whether or not a new database should be created when a database does not exist
mode	Specifies the sharing method of the database; MC_DIR_PRIVATE_ACCESS, MC_DIR_SHARED_ACCESS, or MC_DIR_SYSTEM_ACCESS can be selected

Return Value

Pass	Database ID (an integer that is larger than 0)
Fail	
M_E_NOENT	Issued when create is False, and no database exists
M_E_INVALID	Issued when create is True, recordSize is 0 or a negative number, and dataBaseName is null, and if the mode is not any one of the abovementioned three conditions
M_E_ERROR	Issued when the database cannot be opened due to other reasons

Side Effects

None

Reference Item

MC_dbCloseDataBase, MC_fsOpen

● MC_dbCloseDataBase**Prototype**

M_Int32 MC_dbCloseDataBase(M_Int32 dbId)

Description

Closes a database

Parameters

dbId Database ID

Return Value

Pass

0

Fail

M_E_BADFD

Issued when the Database ID is invalid

M_E_ERROR

Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbOpenDataBase

● MC_dbDeleteDataBase

Prototype

M_Int32 MC_dbDeleteDataBase(M_Char* dataBaseName, M_Int32 mode)

Description

This function deletes a database.

Only databases that can be accessed by the program can be deleted. An open database cannot be deleted.

Parameters

dataBaseName	Name of the database to be deleted
mode	Represents the type of database that can be deleted among databases with varying access rights; MC_DIR_PRIVATE_ACCESS, MC_DIR_SHARED_ACCESS, or MC_DIR_SYSTEM_ACCESS can be selected

Return Value

Pass	0
Fail	
M_E_NOENT	Issued in the absence of a database
M_E_INUSE	Issued when the database is not closed
M_E_INVALID	Issued when dataBaseName is null, or the mode is any one of the abovementioned three conditions
M_E_ACCESS	Issued when trying to delete a database for which the user has no access right
M_E_ERROR	Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbCloseDataBase, MC_fsOpen

● MC_dbInsertRecord

Prototype

M_Int32 MC_dbInsertRecord(M_Int32 dbId, M_Byte* buf, M_Int32 len)

Description

This function adds a new record to the database.

Data stored in the buffer are stored as a record in the database. When the size of a buffer to be stored is smaller than the size of the record specified when the database was created, garbage may be stored in the remaining area.

MC_dbSelectRecord reads data by record size as a unit. Therefore, if data that is smaller than the record size is stored, garbage value may follow the actual data in the character array read thereafter.

Parameters

dbId	ID of a database where a record is added
buf	Buffer where data are stored
len	Size of data stored

Return Value

Pass	Record ID (an integer larger than 0)
Fail	
M_E_BADFD	Issued when the Database ID is invalid
M_E_DATABIG	Issued when data bigger than the size of a record in the database is stored
M_E_INVALID	Issued when len is 0 or a negative number, or the buf is null
M_E_ERROR	Issued when the data cannot be stored

Side Effects

None

Reference Item

MC_dbSelectRecord, MC_dbUpdateRecord, MC_dbDeleteRecord

● MC_dbSelectRecord

Prototype

M_Int32 MC_dbSelectRecord(M_Int32 dbld, M_Int32 recld, M_Byte* buf, M_Int32 len)

Description

This function returns the data stored in a specific Record ID.

The data read are copied to a buffer and returned. When the content of the buffer is changed after this function is called, the record stored in the database will not be changed. The size of the buffer should be bigger than or the same as a record in the relevant database. The size of a record in the database can be obtained using MC_dbGetRecordSize.

Parameters

dbld	Database ID
recl	Record ID
buf	Buffer that will receive the data
len	Length of the buffer

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the Database ID is invalid
	M_E_BADRECID	Issued when the relevant Record ID does not exist
	M_E_INVALID	Issued when len is 0 or a negative number, or when buf is null
	M_E_SHORTBUF	Issued when the given buffer is smaller than the record size
	M_E_ERROR	Issued when the record cannot be read due to other reasons

Side Effects

None

Reference Item

MC_dbInsertRecord, MC_dbUpdateRecord, MC_dbDeleteRecord

● MC_dbUpdateRecord

Prototype

M_Int32 MC_dbUpdateRecord(M_Int32 dbld, M_Int32 recld, M_Byte* buf, M_Int32 len)

Description

Updates the data content of a specific record; this function updates the content stored in the relevant Record ID with a new content

Parameters

dbld	Database ID
recl	Record ID to be used for updating the data content
buf	Buffer containing data to be stored
len	Size of the buffer

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the Database ID is invalid
	M_E_BADRECID	Issued when the Record ID does not exist
	M_E_DATABIG	Issued when the data to be updated is bigger than the record size specified when the database was created
	M_E_INVALID	Issued when len is 0 or a negative number, or when buf is null
	M_E_ERROR	Issued when the record cannot be updated due to other reasons

Side Effects

None

Reference Item

MC_dbInsertRecord, MC_dbSelectRecord, MC_dbDeleteRecord

● MC_dbDeleteRecord**Prototype**

M_Int32 MC_dbDeleteRecord(M_Int32 dbId, M_Int32 recId)

Description

This function deletes a record from the database.

A record corresponding to a Record ID received as a parameter is deleted.

Parameters

dbId	Database ID
recId	Record ID to be deleted

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the Database ID is invalid
	M_E_BADRECID	Issued when the Record ID does not exist
	M_E_ERROR	Issued when the record cannot be deleted due to other reasons

Side Effects

None

Reference Item

MC_dbInsertRecord, MC_dbSelectRecord, MC_dbUpdateRecord

● MC_dbListRecords

Prototype

M_Int32 MC_dbListRecords(M_Int32 dbID, M_Int32* buf, M_Int32 len)

Description

This function returns the Record IDs of records stored in the database.

Record IDs of records stored in the database are stored in an integer array received as a parameter and returned. The sequence of the Record IDs that are returned may differ from the sequence of actual records that are stored.

Parameters

dbId	Database ID
buf	Buffer where the received Record IDs are stored
len	Size of the buffer

Return Value

Pass	Number of records	
Fail	M_E_BADFD	Issued when the Database ID is invalid
	M_E_INVALID	Issued when len is 0 or a negative number, or when buf is null
	M_E_SHORTBUF	Issued when the buffer that will receive the Record ID is too small
	M_E_ERROR	Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbSortRecords

● MC_dbSortRecords

Prototype

```
M_Int32 MC_dbSortRecords(M_Int32 dbID, M_Int32* buf, M_Int32 len,
M_Int32 ((*compare)(const void *, const void *), M_Int32 (*filer)(const
void *))
```

Description

This function returns sorted Record IDs stored in the database.

Each record is compared by byte and sorted.

For example, when the record size of a certain database record is 2 and 4, such records are stored as shown below:

```
=====
| Record ID | Data |
=====
| 0 | 0x33 0x22 |
| 1 | 0x11 0x22 |
| 2 | 0x22 0xff |
| 3 | 0x11 0xff |
=====
```

*

When this function is called, Record IDs are stored in the integer buffer received as a parameter in the sequence of 1, 3, 2, 0, and 4. The number of records) is returned when the two parameters, i.e., compare and filter, are put as null.

The function by which the program developer specifies a sorting method or limits the number of records to be used for sorting can be implemented using the parameters of this function.

A method that determines the sequence of sorting for two records can be specified by implementing compare. Parameters to be received by compare are records stored in the database. This function should compare two records and return an integer that is larger than 0 when the record received as the first parameter precedes the record received as the second parameter, an integer that is smaller than 0 when the record received as the first parameter follows the record received as the second parameter, and 0 when the

sequence of the two is identical.

Filter is a function that determines whether or not a specific record will be included in sorting. This function should return an integer that is larger than 0 to include the record and return 0 to exclude the record from sorting. Parameters received by filter are also records.

Even when this function is called, the structure of the database stored is not changed.

Parameters

dbId	Database ID
buf	Buffer where the received Record IDs are stored
len	Size of the buffer
compare	Function pointer used to compare two records; when null, data will be sorted in ascending order by byte as shown in the above example
filter	Function that limits the number of records to be used for sorting; in case of a null value, all records are used in sorting

Return Value

Pass	Number of records
Fail	
M_E_BADFD	Issued when the Database ID is invalid
M_E_INVALID	Issued when len is 0 or a negative number, or when buf is null
M_E_SHORTBUF	Issued when the buffer that will receive Record IDs is too small
M_E_ERROR	Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbListRecords

● MC_dbGetAccessMode

Prototype

M_Int32 MC_dbGetAccessMode(M_Char* dataBaseName)

Description

This function returns the access right to a database. The access right to a database created by the program, a database that can be shared, and a database used by the system is identified, provided that such access right is limited to databases that can be accessed by the program.

Parameters

dataBaseName Name of a database for which the access right is identified

Return Value

Pass

DIR_PRIVATE_ACCESS, DIR_SHARED_ACCESS, or
DIR_SYSTEM_ACCESS

Fail

M_E_NOENT Issued when the database does not exist
M_E_INVALID Issued when dataBaseName is null
M_E_ERROR Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbOpenDataBase, MC_fsOpen

- **MC_dbGetNumberOfRecords**

Prototype

M_Int32 MC_dbGetNumberOfRecords(M_Int32 dbId)

Description

Returns the number of records that are stored in the database

Parameters

dbId Database ID

Return Value

Pass

Number of records

Fail

M_E_BADFD

Issued when the Database ID is invalid

M_E_ERROR

Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

● MC_dbGetRecordSize**Prototype*****M_Int32 MC_dbGetRecordSize(M_Int32 dbId)*****Description**

Returns the size of one record in the database

Parameters

dbId Database ID

Return Value

Pass

Record size (byte)

Fail

M_E_BADFD

Issued when the Database ID is invalid

M_E_ERROR

Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbOpenDataBase

● MC_dbListDataBases

Prototype

M_Int32 MC_dbListDataBases(M_Byte* buf, M_Int32 len)

Description

This function returns an array of database names. The name of a database that can be accessed by the program from among databases created with MC_DIR_PRIVATE_ACCESS, MC_DIR_SHARED_ACCESS, MC_DIR_SYSTEM_ACCESS, is returned. Each database that is returned through the buffer is distinguished by null (\0), and the end of the list is represented by two consecutive nulls.

Parameters

buf Buffer where the database name is to be stored
len Size of the buffer

Return Value

Pass		Number of databases (regarded as a pass even when there is no database)
Fail	M_E_SHORTBUF	Issued when the buffer is too small to receive an array
	M_E_INVALID	Issued when len is 0 or a negative number, or when buf is null
	M_E_ERROR	Issued in case of failure due to other reasons

Side Effects

None

Reference Item

MC_dbOpenDataBase, MC_fsOpen

2.5. Network

Modules are related to the TCP/IP Internet communication. These include APIs that are related to Internet access, TCP/UDP socket, and HTTP connection.

All network I/O-related APIs are non-blocking APIs. To receive events that occur on I/O, a callback function should be registered.

Data communication is enabled for an application after Internet access is allowed by calling MC_netConnect() through all socket-related APIs. When Internet access is terminated through MC_netSocketClose(), all socket APIs return M_E_NOTCONN errors, and all callback functions registered in sockets are deleted and no longer called. Internet access through MC_netConnect() should be implemented by the application as a unit; Internet access by one application does not affect other applications. For example, even when Internet access is enabled through MC_netConnect(), other applications should call MC_netConnect() for Internet access on their own.

The platform supports TCP and UDP sockets. Support for TCP server socket is an option, although a client function is required. While several TCP sockets and UDP sockets can be supported concurrently, the number that can be supported concurrently varies depending on the platform. M_E_NOSPACE returned by MC_netSocket() means that the socket can no longer be opened. When terminating Internet access by calling the MC_netClose() function, all callback functions registered through network APIs in the program are no longer called, and the platform terminates all sockets created by the program.

- **MC_AF_INET**

Prototype

#define MC_AF_INET 2

Description

A constant that indicates an Internet domain; the value is 2

- **MC_SOCKET_STREAM**

Prototype

#define MC_SOCKET_STREAM 1

Description

A value that specifies the TCP socket type; the value is 1

- **MC_SOCKET_DGRAM**

Prototype

#define MC_SOCKET_DGRAM 2

Description

A value that specifies the UDP socket type; the value is 2

● NETCONNECTCB**Prototype**

typedef void (*NETCONNECTCB)(M_Int32 error, void *param)

Description

Callback function that is registered in the MC_netConnect function

Parameters

error Pass: 0; Fail: M_E_ERROR

param Callback parameter that is passed when calling MC_netConnect()

Return Value

None

Side Effects

None

Reference Item

MC_netConnect

● NETSOCKCONNECTCB

Prototype

```
typedef void(*NETSOCKCONNECTCB)(M_Int32 fd, M_Int32 error, void  
*param)
```

Description

Callback function that is registered in the MC_netConnect function

Parameters

fd	Socket ID
error	0 : Pass; Fail: M_E_ERROR
param	Callback parameter that is passed when calling MC_netSocketConnect()

Return Value

None

Side Effects

None

Reference Item

MC_netSocketConnect

● NETSOCKACCEPTCB

Prototype

```
typedef void(*NETSOCKACCEPTCB)(M_Int32 sd, M_Int32 fd, M_Int32  
error, void *param)
```

Description

Callback function that is registered in the MC_netSocketAccept function

Parameters

sd	Server socket ID
fd	Socket ID connected to the client
error	0 : Pass; Fail: M_E_ERROR
param	Callback parameter that is passed when calling MC_netSocketAccept()

Return Value

None

Side Effects

None

Reference Item

MC_netSocketAccept

● NETSOCKREADCB

Prototype

```
typedef void (*NETSOCKREADCB)(M_Int32 fd, M_Int32 error, void  
*param)
```

Description

Callback function that is registered in the MC_netSetReadCB function

Parameters

fd Socket ID

error 0: Pass; Fail: M_E_ERROR

param Callback parameter that is passed when calling MC_netSetReadCB()

Return Value

None

Side Effects

None

Reference Item

MC_netSetReadCB

● NETSOCKWRITECB

Prototype

```
typedef void(*NETSOCKWRITECB)(M_Int32 fd, M_Int32 error, void  
*param)
```

Description

Callback function that is registered in the MC_netSetWriteCB function

Parameters

fd Socket ID

error Pass: 0; Fail: M_E_ERROR

param Callback parameter that is passed when calling MC_netSetWriteCB()

Return Value

None

Side Effects

None

Reference Item

MC_netSetWriteCB

● NETHOSTADDRCB**Prototype**

typedef void(*NETHOSTADDRCB)(M_Int32 addr, void *param)

Description

This is a Callback function that is registered with MC_netGetHostAddr(). During passing, the IP address value is transmitted as addr instead of -1. The value for this address is network byte ordering.

Parameters

addr IP address

param Callback parameter that is passed when calling MC_netGetHostAddr()

Return Value

None

Side Effects

None

Reference Item

MC_netGetHostAddr

● NETHHTPCB

Prototype

```
typedef void(*NETHHTPCB)(M_Int32 fd, M_Int32 sd, M_Int32 error, void  
*param)
```

Description

This is a callback function that is registered in MC_netHttpConnect() and called when a response is received from the HTTP server, or in case of an error in communication. When a response is received from the server, the point when the parsing of the HTTP header field among server responses is completed is the time when this function is called. Therefore, other functions operate normally only after this function is called.

MC_netHttpGetResponseCode(), MC_netHttpGetResponseMessage(),
MC_netHttpGetHeaderField(), MC_netHttpGetLength(),
MC_netHttpGetType(), MC_netHttpGetEncoding().

In addition, the content transmitted by the server can be read through sd Socket ID, a parameter that is transmitted through this function. In the absence of content transmitted by the server, -1 is transmitted as Socket ID. When Socket ID is not -1, there is no need to call MC_netSocketConnect() since the socket is already connected to the server. When MC_netHttpClose() is called, the abovementioned Socket ID becomes meaningless, and all I/O attempts made through this socket fail. M_E_ERROR transmitted by this function means that a problem occurred during communication. All abovementioned functions will return M_E_ERROR.

Parameters

fd HTTP ID
sd Socket ID
error Pass: 0; Fail: M_E_ERROR
param Callback parameter that is passed when calling MC_netHttpConnect()

Return Value

None

Side Effects

None

Reference Item

MC_netHttpConnect

● **MC_netConnect**

Prototype

*M_Int32 MC_net Connect(NETCONNECTCB cb, void *param)*

Description

This function makes an attempt at Internet access.

TCP/IP communication is enabled only when this function is called for Internet access before using the socket. This is the function that should be called by the application. In other words, even when another application calls this function, and Internet access is allowed, the socket cannot be used unless this function is called by the present application. When 0 is returned, this function indicates whether or not the connection was successfully made through the callback function to be registered. When the application using this function has already called this function, or another application is attempting Internet access, M_E_INPROGRESS is returned. In case Internet access is already allowed for the application that calls this function, an error value of M_E_ISCONN is returned. When this function returns an error value, callback registered in this function is not called.

Parameters

cb Callback function that is called in case of successful or failed attempt at Internet access

Return Value

Pass	0	
Fail	M_E_INPROGRESS	Issued when an attempt at Internet access is made by this application or another application
	M_E_ISCONN	Issued when Internet access is already allowed for this application
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

- **MC_netClose**

Prototype

void MC_netClose(void)

Description

This is a blocking function that terminates Internet access.

When this function is called, Internet communication is rendered impossible. Accordingly, other sockets that are not terminated when this function is called within the application are automatically terminated, together with HTTP IDs that are created through MC_netHttpOpne(). In addition, all callback functions registered through network APIs will not be called. Since this is a function that is called by the application, it does not affect other applications using the socket after this function is called.

Return Value

None

Side Effects

None

Reference Item

None

● MC_netSocket

Prototype

M_Int32 MC_netSocket(M_Int32 domain, M_Int32 type)

Description

This function creates a socket for TCP or UDP communication. Nonetheless, it is necessary to connect to the Internet using MC_netConnect() before this function is called.

Parameters

domain	Communication domain (AF_INET; in case of an Internet socket)
type	MC_SOCKET_STREAM or MC_SOCKET_DGRAM

Return Value

Pass	Socket ID
Fail	
M_E_NOTSUP	Issued when the value is not supported by domain or type
M_E_NOTCONN	Issued when Internet access is not allowed
M_E_NOSPACE	Issued when the socket can no longer be created
M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_netSocketConnect

Prototype

M_Int32 MC_netSocketConnect(M_Int32 fd, M_Int32 addr, M_Int16 port, NETSOCKCONNECTCB cb, void *param)

Description

This function makes an attempt to connect to the server using a TCP socket.

Parameters used such as IP address (addr) and port number (port) should be network byte ordering. When the callback functions to be registered with this function are null, this function returns M_E_INVALID and fails to play any role. When this function returns 0, the function that was registered as a callback is called. In case MC_netSocketClose() is called before the callback function registered with this function is called, the callback function is not called.

Parameters

fd	Socket ID
addr	Server address
port	Port number of the server
cb	Callback function that is called when a connection attempt succeeds or fails
param	Value transmitted when a callback function is called

Return Value

Pass	0	
Fail	M_E_NOTSUP	Issued when the socket does not support this function
	M_E_BADFD	Issued when the Socket ID is invalid
	M_E_NOTCONN	Issued when Internet access is not possible
	M_E_ISCONN	Issued when connection is already made
	M_E_INPROGRESS	Issued when the connection work is already in progress
	M_E_INVALID	Issued when the address is invalid, or the callback function is null
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netSocketClose

● MC_netSocketWrite

Prototype

M_Int32 MC_netSocketWrite(M_Int32 fd, M_Byte* buf, M_Int32 len)

Description

This function sends data with TCP socket.

M_E_WOULDBLOCK returned by this function means that data cannot be transmitted due to an internal factor of the system. In this case, this function is called again within the callback function registered through MC_netSetWriteCB() to transmit the data. Called within the callback function, this function can return M_E_WOULDBLOCK.

Parameters

fd	Socket ID
buf	Data buffer
len	Length of data to be sent

Return Value

Pass	Length of data sent (a value that is larger than 0)	
Fail	M_E_WOULDBLOCK	Issued when the data cannot be transmitted immediately due to an internal factor of the system
	M_E_BADFD	Issued when the Socket ID is invalid
	M_E_NOTCONN	Issued when the connection between the server and the socket is terminated, or Internet access is not possible
	M_E_INVALID	Issued when the buffer or buffer length is invalid

Side Effects

None

Reference Item

MC_netSocketWriteCB

● MC_netSocketRead

Prototype

M_Int32 MC_netSocketRead(M_Int32 fd, M_Byte* buf, M_Int32 len)

Description

This function reads data through the TCP socket.

M_E_WOULDBLOCK returned by this function means that data cannot be read due to an internal factor of the system. In this case, this function is called again within the callback function that was registered through MC_netSetReadCB() to read the data. Called within the callback function, this function can return M_E_WOULDBLOCK.

Parameters

fd	Socket ID
buf	Data buffer
len	Length of data to be read

Return Value

Pass	Length of data read (a value that is larger than 0)	
Fail	M_E_WOULDBLOCK	Issued when the data cannot be read since it has not yet arrived
	M_E_NOTCONN	Issued when the connection between the server and the socket is terminated, or Internet access is not possible
	M_E_BADFD	Issued when the Socket ID is invalid
	M_E_INVALID	Issued when the buffer or buffer length is invalid

Side Effects

None

Reference Item

MC_netSetReadCB

- **MC_netSocketClose**

Prototype

M_Int32 MC_netSocketClose(M_Int32 fd)

Description

This is a blocking function that closes a socket. When Internet access was closed by calling MC_netClose() before calling this function, the socket is already closed. Nonetheless, this function returns Pass. When this function is called, all callback functions registered in the socket are deleted, and callbacks are not called.

Parameters

fd Socket ID

Return Value

Pass

0

Fail

M_E_BADFD

Issued when the Socket ID is invalid

Side Effects

None

Reference Item

MC_netClose

● **MC_netSocketBind**

Prototype

M_Int32 MC_netSocketBind(M_Int32 fd, M_Uint32 addr, M_Uint16 port)

Description

Specifies a specific port to the socket

Parameters

- fd Socket ID
- port Local port number (network byte ordering)

Return Value

- Pass
 - 0
- Fail
 - M_E_INVALID Issued when a specific port is already specified for a different socket
 - M_E_NOTCONN Issued when Internet access is not possible
 - M_E_BADFD Issued when the Socket ID is invalid

Side Effects

None

Reference Item

None

- **MC_netGetMaxPacketLength**

Prototype

M_Int32 MC_netGetMaxPacketLength(void)

Description

Returns the maximum length of a packet that can be sent through UDP

Return Value

Maximum length of a packet

Side Effects

None

Reference Item

None

● MC_netSocketSendTo

Prototype

M_Int32 MC_netSocketSendTo(M_Int32 fd, M_Byte* buf, M_Int32 len, M_Uint32 addr, M_Uint16 port)

Description

Sends data through a UDP socket

Data length to be transmitted cannot exceed the value returned by MC_netGetMaxPacketLength(). Parameters used, i.e., IP address (addr) and port number (port), should be network byte ordering. M_E_WOULDBLOCK returned by this function means that data cannot be transmitted due to an internal factor of the system. In this case, this function is called again within the callback function registered through MC_netSetWriteCB() to transmit the data. Called within the callback, this function can return M_E_WOULDBLOCK again.

Parameters

fd	Socket ID
buf	Data buffer
len	Length of the data
addr	IP address of the opposite party (network byte ordering)
port	Port number of the opposite party (network byte ordering)

Return Value

Pass	Length of data sent (a value that is larger than 0)	
Fail	M_E_NOTSUP	Issued when the socket does not support this function
	M_E_WOULDBLOCK	Issued when the data cannot be transmitted due to an internal factor of the system
	M_E_BADFD	Issued when the Socket ID is invalid
	M_E_NOTCONN	Issued when Internet access is not possible
	M_E_INVALID	Issued when the address, buffer, or buffer length is invalid

Side Effects

None

Reference Item

MC_netGetMaxPacketLength, MC_netSetWriteCB

● MC_netSocketRcvFrom

Prototype

M_Int32 MC_netSocketRcvFrom(M_Int32 fd, M_Byte* buf, M_Int32 len, M_Uint32* addr, M_Uint16* port)

Description

This function reads data through a UDP socket.

Both network byte ordering, the IP address and port number of the opposite party transmitting the data are stored in *addr* and *port*, respectively. *M_E_WOULDBLOCK* returned by this function means that data cannot be read due to an internal factor of the system. In this case, this function is called again within the callback function registered through *MC_netSetReadCB()* to read the data. Called within the callback, this function can return *M_E_WOULDBLOCK* again.

Parameters

<i>fd</i>	Socket ID
<i>buf</i>	Data buffer
<i>len</i>	Length of the data to be read
<i>addr</i>	Buffer where the IP address of the opposite party is stored (network byte ordering)
<i>port</i>	Buffer where the port number of the opposite party is stored (network byte ordering)

Return Value

Pass	Length of data read (a value that is larger than 0)	
Fail	<i>M_E_WOULDBLOCK</i>	Issued when the data cannot be read since it has not yet arrived
	<i>M_E_BADFD</i>	Issued when the Socket ID is invalid
	<i>M_E_NOTCONN</i>	Issued when Internet access is not possible
	<i>M_E_INVALID</i>	Issued when the buffer where the data buffer, buffer length, IP, or port of the opposite party is stored is invalid

Side Effects

None

Reference Item

MC_netSetReadCB

● MC_netGetHostAddr

Prototype

M_Int32 MC_netGetHostAddr(M_Int32 dnsserver, M_Byte *hostname, NETHOSTADDRCB cb, void *param)

Description

This function gets the IP address corresponding to a network host name. Except when this function returns an error, the callback function registered is called when the IP address is obtained, or the attempt failed. Given a dnsserver of -1, this function gets the IP address by accessing the domain name server returned through the MC_knlGetSystemProperty () function.

Parameters

dnsserver	IP address of the domain name server
hostname	Host name
cb	Callback function that is called when an IP address corresponding to a host name is obtained
param	Parameter that is transmitted when the callback function is called

Return Value

Pass	0	
Fail	M_E_INVALID	Issued when the callback function to be registered is null, dnsserver is 0, or parameter hostname is null
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_knlGetSystemProperty

● MC_netSocketAccept

Prototype

M_Int32 MC_netSocketAccept(M_Int32 fd, NETSOCKACCEPTCB cb, void *param)

Description

The TCP server socket returns a socket that is connected to the client. A local port should be specified before this function is called using MC_netSocketBind(). When this function returns M_E_WOULDBLOCK, a callback function that is registered when a socket connected to the client is created is called. On the other hand, when this function returns an error value other than M_E_WOULDBLOCK, this function does not play any role.

Parameters

fd TCP server Socket ID
 cb Callback function
 param Parameter that is transmitted when the callback function is called

Return Value

Pass		Socket ID that is connected to the client
Fail	M_E_INVALID	Issued when the callback function is null
	M_E_WOULDBLOCK	Issued in the absence of a socket connected to the present client
	M_E_NOTSUP	Issued when the socket does not support this function
	M_E_BADFD	Issued when the ID is invalid
	M_E_NOTCONN	Issued when Internet access is not possible
	M_E_ERROR	Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

● MC_netSetReadCB

Prototype

*M_Int32 MC_netSetReadCB(M_Int32 fd, NETSOCKREADCB cb, void *param)*

Description

This function specifies a callback function that is called when data can be read from a socket. The callback registered with this function is called only when MC_netSocketRead() or MC_netSocketRcvFrom() returns M_E_WOULDBLOCK. The callback function specified here is called only once. Therefore, calling this callback function when data can be read requires calling this function again and registering the callback. Likewise, calling this function before the callback function is called causes the previous callback function to be replaced by a new callback function. When the callback function is null, this function removes the previously registered callback function. In addition, if it is possible to read data from the socket when this function is called, the callback function is called.

Parameters

- fd Socket ID
- cb Callback function
- param Callback parameter that is transmitted when the callback function is called

Return Value

- Pass
 - 0
- Fail
 - M_E_NOTCONN Issued when Internet access is not possible
 - M_E_BADF Issued when the Socket ID is invalid

Side Effects

- None

Reference Item

- None

● MC_netSetWriteCB

Prototype

M_Int32 MC_netSetWriteCB(M_Int32 fd, NETSOCKWRITECB cb, void *param)

Description

This function specifies a callback function that is called when data can be read from a socket. The callback registered through this function is called only when MC_netSocketWrite() or MC_netSocketSendTo() returns M_E_WOULDBLOCK. The callback function specified here is called only once. Therefore, calling this callback function called when data can be transmitted requires calling this function again and registering the callback. Likewise, calling this function before the callback function is called causes the previous callback function to be replaced by a new callback function. When the callback function is null, this function removes the previously registered callback function. In addition, if it is possible to transmit data through the socket when this function is called, the callback function is called.

Parameters

fd	Socket ID
cb	Callback function
param	Callback parameter that is transmitted when the callback function is called

Return Value

Pass	0	
Fail	M_E_NOTCONN	Issued when Internet access is not possible
	M_E_BADF	Issued when the Socket ID is invalid

Side Effects

None

Reference Item

None

● MC_netHttpOpen

Prototype

M_Int32 MC_netHttpOpen(M_Byte* url)

Description

This function creates an HTTP ID. After an HTTP connection ID is created through this function, the following functions are called to set or read required values (when the IP address of an HTTP server is not specified in the URL string, the IP address is obtained from the domain name server; the IP address of the domain name server used for such purpose can be obtained through MC_knlGetSystemProperty()):

```
MC_netHttpSetRequestMethod(),
MC_netHttpGetRequestMethod(),
MC_netHttpSetRequestProperty(),
MC_netHttpGetRequestProperty(),
MC_netHttpSetProxy(),
MC_netHttpGetProxy().
```

Parameters

url - URL string that starts with "http://"

Return Value

Pass	HTTP ID	
Fail	M_E_INVALID	Issued when the URL string is invalid
	M_E_NOSPACE	Issued when an HTTP ID can no longer be created
	M_E_NOTCONN	Issued when Internet access is not allowed
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_knlGetSystemProperty, MC_netHttpSetRequestMethod,
 MC_netHttpGetRequestMethod, MC_netHttpSetRequestProperty,
 MC_netHttpGetRequestProperty, MC_netHttpSetProxy, MC_netHttpGetProxy

● MC_netHttpConnect

Prototype

M_Int32 MC_netHttpConnect(M_Int32 fd, NETHHTPCB cb, void *param)

Description

This function attempts to connect to the server through HTTP. When the MC_netHttpOpen() function is called, actual connection between the server and HTTP is not readily made until this function is called. When the connection is made, or the attempt fails, the callback function registered is called. When this function is called, the following functions return M_E_ERROR:

MC_netHttpSetRequestMethod(),
 MC_netHttpGetRequestMethod(),
 MC_netHttpSetRequestProperty(),
 MC_netHttpGetRequestProperty(),
 MC_netHttpSetProxy(),
 MC_netHttpGetProxy().

Parameters

fd HTTP connection ID
 cb Callback function
 param Parameter that is transmitted when a callback function is called

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_NOTCONN	Issued when Internet access is no longer possible
	M_E_INPROGRESS	Issued when connection to the HTTP server is attempted
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpSetRequestMethod,	MC_netHttpGetRequestMethod,
MC_netHttpSetRequestProperty,	MC_netHttpGetRequestProperty,
MC_netHttpSetProxy, MC_netHttpGetProxy	

● MC_netHttpSetRequestMethod

Prototype

M_Int32 MC_netHttpSetRequestMethod(M_Int32 fd, M_Byte *method, M_Byte *msg, M_Int32 msglen)

Description

This function sets the HTTP request method. The request method is GET, POST, or HEAD. A request method of POST means that the message to be transmitted should be displayed together. In case this function is called after the MC_netHttpConnect() function, M_E_ERROR is returned.

Parameters

fd	HTTP ID
method	Request method string
msg	Message to be transmitted when the request method is POST
msglen	Length of the message

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the request method cannot be identified, or when the message is invalid when setting the POST method
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetRequestMethod

Prototype

M_Int32 MC_netHttpGetRequestMethod(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

This function copies the HTTP request method. When the request method is not set through MC_netHttpSetRequestMethod(), GET is copied as a default value. The final request method string that is stored in the parameter buf returns null characters, and the length to be returned is the length of the string excluding null characters. In case this function is called after the MC_netHttpConnect() function, M_E_ERROR is returned.

Parameters

fd HTTP ID
 buf Buffer where the request method is stored
 len Length of the buffer

Return Value

Pass	Length of the request method string stored in the buffer
Fail	
	M_E_BADFD Issued when the ID is invalid
	M_E_INVALID Issued when the buffer or buffer length is invalid
	M_E_ERROR Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpSetRequestProperty

Prototype

```
M_Int32 MC_netHttpSetRequestProperty(M_Int32 fd, M_Byte *key,
M_Byte *value)
```

Description

Sets the HTTP request property; when this function is called after the MC_netHttpConnect() function, M_E_ERROR is returned

Parameters

fd HTTP ID
key Request property name
value Property value

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetRequestProperty

Prototype

M_Int32 MC_netHttpGetRequestProperty(M_Int32 fd, M_Byte *key, M_Byte *buf, M_Int32 len)

Description

Copies the HTTP request property; when a value corresponding to the request property name is not set, or this function is called after the MC_netHttpConnect() function, M_E_ERROR is returned

Parameters

fd	HTTP ID
key	Request property name
buf	Buffer where the value will be stored
len	Length of the buffer

Return Value

Pass	Length of the property value copied	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the buffer or buffer length is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpSetProxy

Prototype

M_Int32 MC_netHttpSetProxy(M_Int32 fd, M_Int32 proxyhost, M_Int16 proxyport)

Description

Sets the proxy; when this function is called after the MC_netHttpConnect() function, M_E_ERROR is returned

Parameters

fd	HTTP ID
proxyhost	IP address of the proxy host; the value is network byte ordering
proxyport	Proxy port number; the value is stored as network byte ordering

Return Value

Pass	0	
Fail		
	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the proxyhost is 0 or -1
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetProxy

Prototype

M_Int32 MC_netHttpGetProxy(M_Int32 fd, M_Int32 *proxyhost, M_Int16 *proxyport)

Description

Copies the proxy host and port; when the proxy is not specified, or this function is called after the MC_netHttpConnect() function, M_E_ERROR is returned

Parameters

fd	HTTP ID
proxyhost	Buffer where the value for proxyhost is stored; the value is network byte ordering
proxyport	Buffer where the value for the proxyport number is stored; the value is stored as network byte ordering

Return Value

Pass	0
Fail	
	M_E_BADFD Issued when the ID is invalid
	M_E_INVALID Issued when proxyhost or proxyport is null
	M_E_ERROR Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● **MC_netHttpGetResponseCode**

Prototype

M_Int32 MC_netHttpGetResponseCode(M_Int32 fd)

Description

This function returns a response code of the HTTP server. When the response from the server is:

HTTP/1.0 200 OK

HTTP/1.0 404 Not Found

the response code will be 200 and 400, respectively. On the other hand, when this function is called before the callback function set by MC_netHttpConnect() is called, M_E_ERROR is returned.

Parameters

fd HTTP ID

Return Value

Pass

Response code

Fail

M_E_BADFD

Issued when the ID is invalid

M_E_ERROR

Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetResponseMessage

Prototype

```
M_Int32 MC_netHttpGetResponseMessage(M_Int32 fd, M_Byte *buf,
M_Int32 len)
```

Description

This function copies a response message of the HTTP server. When the response from the server is:

HTTP/1.0 200 OK

HTTP/1.0 404 Not Found

the response message will be OK or Not Found. On the other hand, when this function is called before the callback function set by MC_netHttpConnect() is called, M_E_ERROR is returned.

Parameters

fd	HTTP ID
buf	Buffer where the response message is stored
len	Length of the buffer

Return Value

Pass	Length of the message copied						
Fail	<table> <tbody> <tr> <td>M_E_BADFD</td> <td>Issued when the ID is invalid</td> </tr> <tr> <td>M_E_INVALID</td> <td>Issued when the buffer or buffer length is invalid</td> </tr> <tr> <td>M_E_ERROR</td> <td>Issued in case of other errors</td> </tr> </tbody> </table>	M_E_BADFD	Issued when the ID is invalid	M_E_INVALID	Issued when the buffer or buffer length is invalid	M_E_ERROR	Issued in case of other errors
M_E_BADFD	Issued when the ID is invalid						
M_E_INVALID	Issued when the buffer or buffer length is invalid						
M_E_ERROR	Issued in case of other errors						

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetHeaderField

Prototype

M_Int32 MC_netHttpGetHeaderField(M_Int32 fd, M_Byte *name, M_Byte *buf, M_Int32 len)

Description

Copies the header value from among the responses of the HTTP server; when this function is called before the callback function set by MC_netHttpConnect() is called, M_E_ERROR is returned

Parameters

fd	HTTP ID
name	Header name
buf	Buffer where the header value is stored
len	Length of the buffer

Return Value

Pass	Length of the header value copied	
Fail		
	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the buffer or buffer length is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetLength

Prototype

M_Int32 MC_netHttpGetLength(M_Int32 fd)

Description

Returns the length of the content; when this function is called before the callback function set by MC_netHttpConnect() is called, M_E_ERROR is returned

Parameters

fd HTTP ID

Return Value

Pass

Length of the content

Fail

M_E_BADFD

Issued when the ID is invalid

M_E_ERROR

Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetType

Prototype

M_Int32 MC_netHttpGetType(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

Copies the type string of the content; when this function is called before the callback function set by MC_netHttpConnect() is called, M_E_ERROR is returned

Parameters

fd	HTTP ID
buf	Buffer from where the content type string is copied
len	Length of the buffer

Return Value

Pass	Length of the content type string copied	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the buffer or buffer length is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

● MC_netHttpGetEncoding

Prototype

M_Int32 MC_netHttpGetEncoding(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

Copies the encoding string of the content; when this function is called before the callback function set by MC_netHttpConnect() is called, M_E_ERROR is returned

Parameters

fd	HTTP ID
buf	Buffer from where the encoding string of the content is copied
len	Length of the buffer

Return Value

Pass	Length of the encoding string of the content copied	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the buffer or buffer length is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_netHttpConnect

- **MC_netHttpClose**

Prototype

M_Int32 MC_netHttpClose(M_Int32 fd)

Description

Ends the use of the HTTP connection ID; when this function is called before the callback function set by MC_netHttpConnect() is called, the callback function is not called

Parameters

fd HTTP ID

Return Value

Pass

0

Fail

M_E_BADFD

Issued when the ID is invalid

Side Effects

None

Reference Item

None

2.6. Serial Communication

APIs are related to serial communication. There may be several serial equipment, with each equipment accessed by port number 0, 1, 2, etc. The number of serial ports supported by the platform can be obtained by passing MAXSERIALNUM to MC_knlGetSystemProperty(). Except for the serial I/O API MC_srIClose() function, the rest are all non-blocking functions. Therefore, in case I/O cannot be processed when the I/O function is called, M_E_WOULDBLOCK is returned. Likewise, when a callback is registered through the callback registration function, the function is called when I/O is possible, or a serial error event occurs in HAL.

● SRLWRITECB**Prototype**

typedef void(*SRLWRITECB)(M_Int32 fd, M_Int32 error, void *param)

Description

A callback function registered in the MC_srlSetWriteCB function, this function is called when data can be transmitted serially, or a serial error event is transmitted to the platform from HAL.

Parameters

fd Port ID

error Pass: 0; Fail: M_E_ERROR

param Callback parameter to be set when registering a callback function

Return Value

None

Side Effects

None

Reference Item

MC_srlSetWriteCB

● SRLREADCB

Prototype

typedef void(*SRLREADCB)(M_Int32 fd, M_Int32 error, void *param)

Description

A callback function that is registered in the MC_srlSetReadCB function, this function is called when data can be read serially, or a serial error event is transmitted to the platform from HAL.

Parameters

fd Port ID

error Pass: 0; Fail: M_E_ERROR

param Callback parameter to be set when registering a callback function

Return Value

None

Side Effects

None

Reference Item

MC_srlSetReadCB

● MC_srlOpen

Prototype

M_Int32 MC_srlOpen(M_Int32 port, M_Byte* param)

Description

This function opens a serial port ID. The serial port ID returned by this function should be a number that is larger than 0. The port to be specified is controlled by a control string that is transmitted through param. The control string should be made using the following syntaxes:

Control string ::= [Control pair]

Control pair ::= Control key = Control value * (, Control key = Control value)

Control key ::= String that consists of alphabets or alphanumeric values

Control value ::= String that consists of alphabets or alphanumeric values

The control keys and control values defined in this document are shown in the following table:

Control Key	Control Value	Meaning	Default
Baudrate	9600, 19200, 38400, 57600, 115200	Speed	115200
Parity	Even, odd, none	Parity	None
Size	Number	Byte size	8
Flow	hardware, software, none	Flow control	None

Definitions for control keys and control values other than those described above can also be added. For example, when opening a port for flow control of hardware in 8 bit, no parity, and 115200 speed, the control string will be: "baudrate=115200, parity=none, size=8, flow=hardware". In case a control key defined above is omitted from the control string when opening a serial port, the default value will be set.

Parameters

port Port number (0 ~(maximum number of ports - 1))

param Control string

Return Value

Pass

ID return

Fail

M_E_NOTSUP Issued when control for the port number or control string is not supported

M_E_INVALID	Issued when the format of the control string is invalid
M_E_ISCONN	Issued when the port is already in use

Side Effects

None

Reference Item

None

● **MC_srlWrite**

Prototype

M_Int32 MC_srlWrite(M_Int32 fd, M_Uint8 buf, M_Int32 size)*

Description

This function transmits data using a serial port. When data cannot be transmitted immediately due to an internal factor of the system, M_E_WOULDBLOCK is returned. In this case, this function is called again within the callback that was registered through MC_srlSetWriteCB to transmit the data. Called within the callback function, this function can return M_E_WOULDBLOCK once again.

Parameters

fd	ID
buf	Data buffer
size	Size of the data

Return Value

Pass	Size of the data written
Fail	
	M_E_WOULDBLOCK Issued when the current data cannot be written
	M_E_BADFD Issued when the ID is invalid
	M_E_INVALID Issued when the buffer or buffer length is invalid

Side Effects

None

Reference Item

MC_srlSetWriteCB

● MC_srlSetReadCB

Prototype

*M_Int32 MC_srlSetReadCB(M_Int32 fd, SRLREADCB cb, void *param)*

Description

This function registers a callback function that is called when data can be read using a serial port. When the callback function is called, param is transmitted as API of the callback function. In case the callback function to be registered is null, the callback registered using this function is deleted. When this function returns an error, callback is not called. On the other hand, when data can be read when this function is called from the serial port, the callback function is called.

Parameters

- fd Port ID
- cb Callback function
- param Value transmitted when the callback function is called

Return Value

- Pass
 - 0
- Fail
 - M_E_BADFD Issued when the ID is invalid
 - M_E_ERROR Issued in case of other errors

Side Effects

- None

Reference Item

- None

● **MC_srlSetWriteCB**

Prototype

M_Int32 MC_srlSetWriteCB(M_Int32 fd, SRLWRITECB cb, void *param)

Description

This function registers a callback function that is called when data can be transmitted using a serial port. When the callback function is called, param is transmitted as API of the callback function. In case the callback function to be registered is null, the callback function registered using this function is deleted. When this function returns an error, the callback function is not called. On the other hand, when data can be transmitted upon calling this function from the serial port, the callback function is called.

Parameters

- fd Port ID
- cb Callback function
- param Value transmitted when the callback function is called

Return Value

- Pass
 - 0
- Fail
 - M_E_BADFD Issued when the ID is invalid
 - M_E_ERROR Issued in case of other errors

Side Effects

- None

Reference Item

- None

● MC_srlClose

Prototype

M_Int32 MC_srlClose(M_Int32 fd)

Description

Ends the use of a serial port; when ending the use of a serial port, all the callback functions registered in the serial port are deleted and will no longer be called

Parameters

fd Port ID

Return Value

Pass

0

Fail

M_E_BADFD

Issued when the ID is invalid

Side Effects

None

Reference Item

None

2.7. User Interface Component

Included in the user interface component are text box, date/time component, menu component, label component, and list component.

● Creation and Extinction of Component

Creating components require a component class structure. A component class structure is a pre-defined structure whose values are already determined. This structure can be obtained using MC_uicGetClass. A component class structure varies depending on the component, and it can be created by calling the MC_uicCreate function using this component class structure as a parameter. A created component can be destroyed using the MC_uicDestroy function.

● Coordinate System

The position and size of a component follows the coordinate system on the display. The offset (0, 0) is located on the upper left, and the coordinate value increases in pixel as it moves to the right and downward.

● Handling of Event

User interface components provided by C do not supply containers. Therefore, works carried out by a container should be performed in each program concerned. In other words, the program should call a function that handles each event for the relevant component.

The program should call the MC_uicHandleEvent function using each component created within the handleCletEvent function of Clet as parameters. MC_uicHandleEvent handles an event as required and returns the handling status for the event. The MC_uicPaint function should be called within the paintClet function of Clet for each component.

● Callback Function and Event Handler Function

When an event is handled by MC_uicHandleEvent, the user calls the event handler function initially specified by the MC_uicSetEventHandler function. Depending on the value returned by the function called, the MC_uicHandleEvent function determines whether to continue handling the event. In case the event is handled,

MC_uicHandleEvent handles the event and calls MC_uicCallbackProc for the relevant event. Since the availability of the callback function for an event varies depending on the component, be sure to check the relevant documents for the component.

● Text Box Component

The text box component is a component that inherits MC_UicComponent, sends a string to the user, and allows the string to be changed. The string that can be put in the text box can be determined using MC_uicGetTextSize. By default, the maximum value that can be stored is 256 bytes, but this can be changed using the MC_uicSetTextSize function.

The available functions are as follows:

MC_uicInsertText

MC_uicDeleteText

MC_uicGetMaxTextSize

MC_uicSetMaxTextSize

MC_uicGetTextSize

MC_uicGetText

Additional functions that are available to the component:

The following are the available callback functions:

MC_UIC_CHANGE_CALLBACK: Used when an internal string has been changed

MC_UIC_KEY_CALLBACK: Used when a key is pressed

Additional callback functions that are available to the component:

● Date/Time Component

The date/time component is a function that inherits MC_UicComponent, shows the date and time to the user, and receives an input for the date and time. Date and time can be specified using the MC_uicSetTime function. When it is not specified, it is set to the date and time when the component was created. The date/time component can print the date or time or both on the display using the MC_uicSetTimeMask function.

The following are the available functions:

MC_uicSetTimeMask

MC_uicSetTime

MC_uicGetTime

MC_uicSetTimeLong

Additional functions that are available to the component:

The following are the available callback functions:

MC_UIC_CHANGE_CALLBACK: Used when the internal time has been changed

Additional callback functions that are available to the component:

- **Menu Component**

The menu component inherits MC_UicComponent, prints menu on the display, and receives inputs from the user. Currently selected menu items can be changed using arrow keys by moving the key upward/downward or to the left or to the right.

The following are available functions:

MC_uicAddMenuItem

MC_uicGetMenuItem

MC_uicRemoveMenuItem

MC_uicSetActiveMenuItem

MC_uicGetActiveMenuItem

Additional functions that are available to the component:

The following are available callback functions:

MC_UIC_CHANGE_CALLBACK: Used when an internally selected menu has been changed

Additional callback functions that are available to the component:

- **Label Component**

The label component is a function that inherits MC_UicComponent and prints a string on the display. It also changes lines to enable the string to fit its size. Unlike other components, however, this component does not handle events.

The following are available functions:

MC_uicSetLabel

- **List Component**

The list component inherits MC_UicComponent, prints a list on the display, and receives inputs from the user. Menu items can be changed using the menu upward/downward key.

The following are available functions:

MC_uicAddListItem

MC_uicGetListItem

MC_uicRemoveListItem

MC_uicSetActiveListItem

MC_uicGetActiveListItem

Additional functions that are available to the component:

The following are available callback functions:

MC_UIC_CHANGE_CALLBACK: Used when an internally selected menu has been changed

Additional callback functions that are available to the component:

- **Component**

Component is the most basic component that does not handle events.

The following are available functions:

MC_uicDestroy

MC_uicRepaint

MC_uicPaint

MC_uicGetClassName

MC_uicIsInstance

MC_uicHandleEvent

MC_uicConfigure

MC_uicGetGeometry

MC_uicSetEnable

MC_uicSetExtData

MC_uicGetExtData

MC_uicSetCallback

MC_uicSetEventHandler

MC_uicSetFont

MC_uicGetFont

The following are available callback functions:

MC_UIC_SELECT_CALLBACK: Used when the user selected an item by pressing SELECT

MC_UIC_PAINT_CALLBACK: Used when a component is painted

MC_UIC_DESTROY_CALLBACK: Used when a component is destroyed

- **MC_UicComponent**

Prototype

```
typedef M_Int32 MC_UicComponent
```

Description

Component ID

- **MC_UicClass**

Prototype

typedef M_Int32 MC_UicClass

Description

Component class structure ID

- **MC_UicApplicationContext**

Prototype

typedef M_Int32 MC_UicApplicationContext

Description

Program context ID

● MC_UicCallbackProc**Prototype**

```
typedef void(*MC_UicCallbackProc)(MC_UicComponent cc, void*  
serverData, M_Int32 clientData)
```

Description

A type of callback function

Parameters

cc	Component that calls a callback function
serverData	Data passed by the user; varies depending on the user
clientData	callback function

Side Effects

None

Reference Item

None

- **MC_UicEventHandlerProc**

Prototype

```
typedef   M_Int32(*MC_UicEventHandlerProc)(MC_UicComponent   cc,  
      M_Int32 type, M_Int32 param1, M_Int32 param2)
```

Description

A type of event handler function

Parameters

<i>cc</i>	Component that calls the event handler
<i>type</i>	Event type
<i>param1</i>	Event parameter 1
<i>param2</i>	Event parameter 2

Return Value

Event handling status (returns 1 when the event is handled; otherwise, 0 is returned)

Side Effects

None

Reference Item

None

- **MC_UIC_MENU_COMPONENT**

Prototype

#define MC_UIC_MENU_COMPONENT "MenuComponent"

Description

A string of menu component class; this is defined as MenuComponent

- **MC_UIC_DATE_TIME_COMPONENT**

Prototype

```
#define                                MC_UIC_DATE_TIME_COMPONENT  
    "DateTimeComponent"
```

Description

A string of date/time component; this is defined as DateTimeComponent

- **MC_UIC_TEXT_COMPONENT**

Prototype

```
#define MC_UIC_TEXT_COMPONENT      "TextComponent"
```

Description

A string of text component class; this is defined as TextComponent

- **MC_UIC_LABEL_COMPONENT**

Prototype

#define MC_UIC_LABEL_COMPONENT "LabelComponent"

Description

A string of label component class; this is defined as LabelComponent

- **MC_UIC_LIST_COMPONENT**

Prototype

```
#define MC_UIC_LIST_COMPONENT      "ListComponent"
```

Description

A string of list component class; this is defined as ListComponent

- **MC_UIC_DESTROY_CALLBACK**

Prototype

#define MC_UIC_DESTROY_CALLBACK 1

Description

Index 1 of the callback function that is called when the component is destroyed

- **MC_UIC_PAINT_CALLBACK**

Prototype

```
#define MC_UIC_PAINT_CALLBACK 2
```

Description

Index of the callback function that is called when the component is painted; this is defined as constant 2

- **MC_UIC_SELECT_CALLBACK**

Prototype

```
#define MC_UIC_SELECT_CALLBACK    3
```

Description

Index of the callback function that is called when a specific content of the component is selected; this is defined as constant 3

- **MC_UIC_CHANGE_CALLBACK**

Prototype

```
#define MC_UIC_CHANGE_CALLBACK 4
```

Description

Index of the callback function that is called when the content of the component is changed; this is defined as constant 4

- **MC_UIC_KEY_CALLBACK**

Prototype***#define MC_UIC_KEY_CALLBACK 5*****Description**

Index of the callback function that is called when the user pressed a key for the component; this is defined as constant 5

- **MC_GRP_GRAY_TYPE**

Prototype

```
#define MC_GRP_GRAY_TYPE    2
```

Description

A black and white type; this is defined as constant 2

- **MC_GRP_COLOR_TYPE**

Prototype

```
#define MC_GRP_COLOR_TYPE    4
```

Description

A color type; this is defined as constant 4

- **MC_ALIGN_LEFT**

Prototype

```
#define MC_ALIGN_LEFT 0
```

Description

Left alignment of the label component; this is defined as constant 0

- **MC_ALIGN_RIGHT**

Prototype

```
#define MC_ALIGN_RIGHT 1
```

Description

Right alignment of the label component; this is defined as constant 1

- **MC_ALIGN_CENTER**

Prototype

```
#define MC_ALIGN_CENTER 2
```

Description

Center alignment of the label component; this is defined as constant 2

- **MC_uicCreateApplicationContext**

Prototype

MC_UicApplicationContext MC_uicCreateApplicationContext()

Description

Creates the application context

Parameters

None

Return Value

Application context created

Side Effects

None

Reference Item

None

- **MC_uicGetClass**

Prototype

MC_UicClass MC_uicGetClass(M_Uint8* psz)

Description

This function gets a component class structure corresponding to a string. Strings of currently supported component class structures include MC_UIC_MENU_COMPONENT, MC_UIC_DATE_TIME_COMPONENT, MC_UIC_TEXT_COMPONENT, MC_UIC_LABEL_COMPONENT, and MC_UIC_LIST_COMPONENT.

Parameters

[in] psz String representing a component class

Return Value

Component class structure; when the string is not a string of the component class structure for which a parameter (psz) is supported, M_E_ERROR is returned

Side Effects

None

Reference Item

None

● MC_uicCreate

Prototype

MC_UicComponent *MC_uicCreate*(*MC_UicApplicationContext* *pac*,
MC_UicClass *cls*)

Description

This function creates a component from the component class specified. The application context created from MC_uicCreateApplicationContext and class structure brought to the MC_uicGetClass function are passed as parameters.

Parameters

[in] *pac* Application context
[in] *cls* Application context [how is this different from *pac*?]

Return Value

Pass	New component class created
Fail	
	M_E_ERROR Issued when an invalid parameter is used, or in case of other errors
	MC_UIC_E_OUT_OF_MEM Issued in case of insufficient memory

Side Effects

None

Reference Item

MC_uicCreateApplicationContext

- **MC_uicDestroy**

Prototype

void MC_uicDestroy(MC_UicComponent cc)

Description

Destroys a specified component; a deleted cc should no longer be used

Parameters

[in] cc Component to be deleted

Side Effects

None

Reference Item

None

- **MC_uicRepaint**

Prototype

```
void MC_uicRepaint(MC_UicComponent cc, M_Int32 x, M_Int32 y,  
M_Int32 w, M_Int32 h)
```

Description

This function creates a paint event for the specified component. An internal event is created to enable the paintClet function to be called for the specified component area. When w and h are -1, the area from the specified (x, y) coordinates to the lower right part of the component is the area where a paint event occurs.

Parameters

[in]	cc	Component
[in]	x	X coordinate (component coordinate system) of the area within the component
[in]	y	Y coordinate (component coordinate system) of the area within the component
[in]	w	Width of the area within the component
[in]	h	Height of the area within the component

Side Effects

None

Reference Item

MC_grpRepaint, MC_uicPaint

- **MC_uicPaint**

Prototype

void MC_uicPaint(MC_UicComponent cc, MC_GrpContext *pgc)

Description

This function paints a specified component. The content of a component is painted on the frame buffer of the actual display. This function should be called in the paintClet function of the program by component. The clipping area specified by pgc is not changed; instead, part of the content of pgc can be changed depending on the component.

Parameters

[in]	cc	Component
[in]	pgc	Graphic context to be used

Side Effects

None

Reference Item

None

- **MC_uicGetClassName**

Prototype

M_Uint8* MC_uicGetClassName(MC_UicComponent cc)

Description

This function gets the class name of a component. The class name corresponding to a component indicated by cc is received in C string. Such value cannot be changed.

Parameters

[in] cc Component

Return Value

Class name of a component; null is returned as an error value when an invalid parameter is used

Side Effects

None

Reference Item

None

- **MC_uiclsInstance**

Prototype

M_UInt32 MC_uiclsInstance(MC_UicComponent cc, M_UInt8* pcls)

Description

Indicates whether the component is a class that inherited a specified class

Parameters

[in]	cc	Component
[in]	pcls	String of a component class name

Return Value

Returns 1 when the component is an instance of the specified class; otherwise, 0 is returned

Side Effects

None

Reference Item

None

- **MC_uicHandleEvent**

Prototype

M_Int32 MC_uicHandleEvent(MC_UicComponent cc, M_Int32 type, M_Int32 param1, M_Int32 param2)

Description

This function handles events by component.

A specified component is requested to handle the event. When the event is handled, 1 is returned. Otherwise, 0 is returned.

Type, param1, and param2 pass event-related parameters from the handleCletEvent function as they are.

Parameters

[in]	cc	Component
[in]	type	Event type
[in]	param1	Event parameter 1
[in]	param2	Event parameter 2

Return Value

Event handling status within the component

Side Effects

None

Reference Item

None

● MC_uicConfigure**Prototype**

```
void MC_uicConfigure(MC_UicComponent cc, M_Int32 x, M_Int32 y,  
M_Int32 w, M_Int32 h, M_Int32 mask)
```

Description

This function specifies the size and position of a component.

A specified component is placed on a position indicated by (x, y), and its size is determined through width w and height h. When mask and MC_UIC_POS_MASK are not 0, the component is moved to a position indicated by (x, y). Likewise, when mask and MC_UIC_SIZE_MASK are not 0, the component is specified as size (w, h).

When w or h is below 0, this function does not operate.

Parameters

[in]	cc	Component
[in]	x	X coordinate of the component on the display
[in]	y	Y coordinate of the component on the display
[in]	w	Width of the component
[in]	h	Height of the component
[in]	mask	Whether only the size of the component or only the height is changed; MC_UIC_SIZE_MASK or MC_UIC_POS_MASK

Side Effects

None

Reference Item

None

● MC_uicGetGeometry**Prototype**

```
void MC_uicGetGeometry(MC_UicComponent cc, M_Int32* px, M_Int32*  
py, M_Int32* pw, M_Int32* ph)
```

Description

This function gets the size and position of the component.

The position of a specified component is stored in a variable indicated by px and py, and the size, in pw and ph. When one of the parameters is null, the corresponding value is not copied.

Parameters

[in]	cc	Component
[out]	px	Pointer where the x value of the component on the display is stored
[out]	py	Pointer where the y value of the component on the display is stored
[out]	pw	Pointer where the width of the component is stored
[out]	ph	Pointer where the height of the component is stored

Side Effects

None

Reference Item

None

- **MC_uicSetEnable**

Prototype

void MC_uicSetEnable(MC_UicComponent cc, M_Int32 enable)

Description

This function enables or disables a component to receive an input. By default, a component is disabled when it is created (enable is set to False). Therefore, for the component to receive inputs, the component should be enabled using this function.

Parameters

[in]	cc	Component
[in]	enable	When 1, enables the component; when 0, disables the component

Side Effects

None

Reference Item

None

- **MC_uicSetEventHandler**

Prototype

MC_UicEventHandlerProc MC_uicSetEventHandler(MC_UicComponent cc, MC_UicEventHandlerProc handler)

Description

This function specifies an event handler to the component. When an event handler is specified, the event handler is called for all events of the component. Whether events will be handled continuously depends on the result value of the event handler.

Parameters

[in]	cc	Component
[in]	handler	Event handler

Return Value

Previously specified event handler; in the absence of a previously specified event handler, null is returned

Side Effects

None

Reference Item

None

- **MC_uicSetFont**

Prototype

M_Int32 MC_uicSetFont(MC_UicComponent cc, M_Int32 fontid)

Description

Specifies a font for the component using Font ID specified by fontid

Parameters

[in] cc Component

[in] fontid Font ID as the return value of MC_grpGetFont

Return Value

Previously specified Font ID

Side Effects

None

Reference Item

None

- **MC_uicGetFont**

Prototype

M_Int32 MC_uicGetFont(MC_UicComponent cc)

Description

Returns a font of the component

Parameters

[in] cc Component

Return Value

Currently specified Font ID

Side Effects

None

Reference Item

None

- **MC_uicSetFgColor**

Prototype

```
void MC_uicSetFgColor(MC_UicComponent cc, M_Int32 nColor);
```

Description

Sets the foreground color for a component

Parameters

[in]	cc	Component
[in]	nColor	Color value (0 x RRGGBB)

Return Value

None

Side Effects

None

Reference Item

None

- **MC_uicSetBgColor**

Prototype

```
void MC_uicSetBgColor(MC_UicComponent cc, M_Int32 nColor);
```

Description

Sets the background color for a component

Parameters

[in]	cc	Component
[in]	nColor	Color value (0 x RRGGBB)

Return Value

None

Side Effects

None

Reference Item

None

- **MC_uicSetLabel**

Prototype

void MC_uicSetLabel(MC_UicComponent cc, M_Uint8 *psz)

Description

Changes the string for a label component; when the specified component is not a label component, this function does not play any role

Parameters

[in]	cc	Component
[in]	psz	String to be changed

Return Value

None

Side Effects

None

Reference Item

None

- **MC_uicGetLabel**

Prototype

M_Uint8* MC_uicGetLabel(MC_UicComponent cc)

Description

Gets the string for a label component; when the specified component is not a label component, this function does not play any role

Parameters

[in] cc Component

Return Value

Currently specified string

Side Effects

None

Reference Item

None

- **MC_uicSetlabelAlignment**

Prototype

M_Int32 MC_uicSetlabelAlignment(MC_UicComponent cc, M_Int32 align)

Description

Sets the sorting method of the string for a label component; MC_ALIGN_LEFT, MC_ALIGN_RIGHT, or MC_ALIGN_CENTER can be used, which refers to the left alignment, right alignment, and center alignment, respectively (when the specified component is not a label component, this function does not play any role)

Parameters

[in]	cc	Component
[in]	align	Sorting method of the string for a label component; MC_ALIGN_LEFT, MC_ALIGN_RIGHT, or MC_ALIGN_CENTER can be used

Return Value

Receives the previously specified value

Side Effects

None

Reference Item

None

- **MC_uicSetTimeMask**

Prototype

M_Int32 MC_uicSetTimeMask(MC_UicComponent cc, M_Int32 mask)

Description

This function specifies the type for a time component.

When mask and MC_UIC_TIME_MASK are not 0, time is displayed. On the other hand, when mask and MC_UIC_DATE_MASK are not 0, date is displayed.

When the specified component is not a time component, this function does not play any role.

Parameters

[in]	cc	Component
[in]	mask	Mask of a time component, i.e., MC_UIC_TIME_MASK or MC_UIC_DATE_MASK

Return Value

Receives a previously specified value

Side Effects

None

Reference Item

None

- **MC_uicSetTime**

Prototype

void MC_uicSetTime(MC_UicComponent cc, struct tm* pd)

Description

This function specifies the time to a time component.

When the specified component is not a time component, this function does not play any role. Here, pd has the form of a tm structure that is defined in time.h of the standard C library.

Parameters

[in]	cc	Component
[in]	pd	Time to be specified (refer to time.h)

Side Effects

None

Reference Item

None

- **MC_uicSetTimeLong**

Prototype

void MC_uicSetTimeLong(MC_UicComponent cc, time_t time)

Description

This function specifies the time to a time component.

When the specified component is not a time component, this function does not play any role. Here, time has the form of time_t type value obtained in the time () function that is defined in time.h of the standard C library.

Parameters

[in]	cc	Component
[in]	time	Date and time to be specified (refer to time.h)

Side Effects

None

Reference Item

None

- **MC_uicGetTime**

Prototype

void MC_uicGetTime(MC_UicComponet cc, struct tm* pd)

Description

This function gets the time/date from the time component.

This function fills the content to each transmitting pointer. When the pointer indicates null, the corresponding value is not copied.

When the specified component is not a time component, this function does not play any role. Here, the date received has the form of a tm structure that is defined in time.h of the standard C library.

Parameters

[in]	cc	Component
[out]	pd	Data to be received

Side Effects

None

Reference Item

None

● MC_uicAddMenuItem**Prototype**

***M_Int32 MC_uicAddMenuItem(MC_Component cc, M_Uint8* psz,
MC_GrplImage img)***

Description

This function adds a menu item to the menu component.

When the specified component is not a menu component, this function does not play any role.

The image to be passed is internally destroyed when the menu item is deleted.

Parameters

[in]	cc	Component
[in]	psz	String representing the menu item
[in]	img	Image of the menu item

Return Value

Menu index (starts with 0)

Side Effects

None

Reference Item

None

● MC_uicGetMenuItem**Prototype**

***M_Int32 MC_uicGetMenuItem(MC_Component cc, M_Uint32 idx,
M_Uint8* psz, M_Int32 buflen, MC_GrplImage* img)***

Description

Gets a menu item; when the specified component is not a menu component, this function does not play any role

Parameters

[in]	cc	Component
[in]	idx	Index of the menu item to be brought
[out]	psz	Buffer from where the string representing the menu item is copied
[in]	buflen	Size of the buffer
[out]	img	Image of the menu item

Return Value

Pass when the value is 1, fail when the value is below 0

M_E_SHOFTBUF: Issued when the buffer size that is passed as psz is too small

Side Effects

None

Reference Item

None

- **MC_uicRemoveMenuItem**

Prototype

M_Int32 MC_uicRemoveMenuItem(MC_Component cc, M_Uint32 idx)

Description

Removes a menu item that is indicated by idx; when the specified component is not a menu component, this function does not play any role

Parameters

[in]	cc	Component
[in]	idx	Index of the menu item to be removed

Return Value

Pass when the value is 1; fail when the value is 0

Side Effects

None

Reference Item

None

- **MC_uicSetActiveMenuItem**

Prototype

M_Int32 MC_uicSetActiveMenuItem(MC_Component cc, M_Int32 idx)

Description

This function activates a specified menu item. When the specified component is not a menu component, this function does not play any role. On the other hand, when the idx value is out of bounds for a menu item, this function does not play any role. To maintain the unselected status, set idx to -1.

Parameters

[in]	cc	Component
[in]	idx	Index of the menu item

Return Value

Passes the previously activated menu index

Side Effects

None

Reference Item

None

● MC_uicGetActiveMenuItem**Prototype**

M_Int32 MC_uicGetActiveMenuItem(MC_Component cc)

Description

The active menu item index is returned.

When the specified component is not a menu component, this function does not play any role.

Parameters

[in] cc Component

Return Value

Active menu item index

-1: When there is no active menu

Side Effects

None

Reference Item

None

- **MC_uicInsertText**

Prototype

M_Int32 MC_uicInsertText(MC_Component cc, M_Int32 idx, M_Uint8* psz, M_Int32 len)

Description

This function adds a string to the text component.

Starting with the idx part of the text component that is indicated by cc, insert the psz content up to len. When idx is 0, addition occurs at the front; on the other hand, when idx is more than the value passed by M_CuicGetMaxTextSize(), addition occurs at the end.

Since a string is copied internally, the content indicated by psz can be changed after this function is closed.

When the specified component is not a text component, this function does not play any role.

Parameters

[in]	cc	Component
[in]	idx	Position where the string is added; it can be from 0 to the last string
[in]	psz	String to be copied
[in]	len	Length of the string

Return Value

Length of the string copied to the component; in case of insufficient memory, a value that is smaller than len is returned

Side Effects

None

Reference Item

None

- **MC_uicDeleteText**

Prototype

void MC_uicDeleteText(MC_Component cc, M_Int32 idx, M_Int32 len)

Description

This function deletes a specific part of the text component indicated by cc up to len starting from the idx part.

When len is -1, the part from idx to the end of the sentence is deleted.

In case idx is out of bounds for an internal string, or len is smaller than 0, this function does not play any role.

When the specified component is not a text component, this function does not play any role.

Parameters

[in]	cc	Component
[in]	idx	Starting position of the string to be deleted
[in]	len	Length of the string

Side Effects

None

Reference Item

None

- **MC_uicGetMaxTextSize**

Prototype

M_Int32 MC_uicGetMaxTextSize(MC_Component cc)

Description

Gets the maximum string size of the text component; this function returns the maximum string size in byte (when the specified component is not a text component, this function does not play any role)

Parameters

[in] cc Component

Return Value

Maximum string size

Side Effects

None

Reference Item

None

- **MC_uicSetMaxTextSize**

Prototype

M_Int32 MC_uicSetMaxTextSize(MC_Component cc, M_Int32 max)

Description

This function specifies the maximum string size of the text component. The maximum string size is specified in byte.

When the specified component is not a text component, this function does not play any role. The default size is 256 bytes.

If the new internal string defined is smaller than max when calling this function, the content of the internal buffer is cut to the max size.

Parameters

[in]	cc	Component
[in]	max	Maximum string size (byte)

Return Value

Currently existing maximum string size; in case of insufficient memory, M_E_NOMEMORY is returned

Side Effects

None

Reference Item

None

- **MC_uicGetTextSize**

Prototype

M_Int32 MC_uicGetTextSize(MC_Component cc)

Description

Gets the length of the internal string of the text component; when the specified component is not a text component, this function does not play any role

Parameters

[in] cc Component

Return Value

Length of the current string

Side Effects

None

Reference Item

None

- **MC_uicGetText**

Prototype

M_Int32 MC_uicGetText(MC_Component cc, M_Int32 idx, M_Uint8 pszBuf, M_Int32 len)*

Description

This function returns the internal string of the text component.

The internal string of the component indicated by cc is copied to pszBuf up to len starting from the idx part.

Idx should not get out of bounds for a string. An idx that is smaller than 0 is adjusted to 0. When idx is bigger than the string, however, no content is copied, and 0 is returned.

When the specified component is not a text component, this function does not play any role.

Parameters

[in]	cc	Component
[in]	idx	Starting index of the string
[out]	psz	Buffer to be copied
[in]	len	Size of the buffer

Return Value

Length of the string copied

Side Effects

None

Reference Item

None

- **MC_uicAddListItem**

Prototype

M_Int32 MC_uicAddListItem(MC_Component cc, M_Uint8* psz, MC_GrplImage img)

Description

This function adds a list item to the list component.

When the specified component is not a list component, this function does not play any role.

The image passed is internally destroyed when the list item is deleted.

Parameters

[in]	cc	Component
[in]	psz	String representing the list item
[in]	img	Image of the list item

Return Value

List index (starts with 0)

Side Effects

None

Reference Item

None

- **MC_uicGetListItem**

Prototype

M_Int32 MC_uicGetListItem(MC_Component cc, M_UInt32 idx, M_UInt8* psz, M_Int32 buflen, MC_GrplImage* img)

Description

Gets a list item; when the specified component is not a list component, this function does not play any role

Parameters

[in]	cc	Component
[in]	idx	Index of the list item to be brought
[out]	psz	Buffer from where the string representing the list item is copied
[in]	buflen	Size of the buffer
[out]	img	Image of the list item

Return Value

Pass when the value is 1, fail when the value is below 0

M_E_SHOFTBUF: Issued when the buffer size that is passed as psz is too small

Side Effects

None

Reference Item

None

- **MC_uicRemoveListItem**

Prototype

M_Int32 MC_uicRemoveListItem(MC_Component cc, M_Uint32 idx)

Description

Removes a specified list item indicated by idx; when the specified component is not a label component, this function does not play any role

Parameters

[in]	cc	Component
[in]	idx	Index of the list item to be removed

Return Value

Pass	1
Fail	0

Side Effects

None

Reference Item

None

- **MC_uicSetActiveListItem**

Prototype

M_Int32 MC_uicSetActiveListItem(MC_Component cc, M_Int32 idx)

Description

This function activates a specified list item.

When the specified component is not a list component, this function does not play any role. Similarly, when the idx value is out of bounds for a list item, this function does not play any role. To maintain the unselected status, set idx to -1.

Parameters

[in]	cc	Component
[in]	idx	Index of the list item

Return Value

Passes the previously activated list index

Side Effects

None

Reference Item

None

● MC_uicGetActiveListItem**Prototype**

M_Int32 MC_uicGetActiveListItem(MC_Component cc)

Description

Returns an active list item index; when the specified component is not a list component, this function does not play any role

Parameters

[in] cc Component

Return Value

Pass

Active list item index

Fail

-1: When there is no active list

Side Effects

None

Reference Item

None

2.8. Utility

Various utility functions are defined.

- **MC_utilHtonl**

Prototype

M_Int32 MC_utilHtonl(M_Int32 val)

Description

Converts host byte ordering of M_Int32 type value to network byte ordering

Return Value

Integer value

Side Effects

None

Reference Item

None

- **MC_utilHtons**

Prototype

M_Int16 MC_utilHtons(M_Int16 val)

Description

Converts host byte ordering of M_Int16 type value to network byte ordering

Return Value

Integer value

Side Effects

None

Reference Item

None

- **MC_utilNtohI**

Prototype

M_Int32 MC_utilNtohI(M_Int32 val)

Description

Converts network byte ordering of M_Int32 type value to host byte ordering

Return Value

Integer value

Side Effects

None

Reference Item

None

- **MC_utilNtohs**

Prototype

M_Int16 MC_utilNtohs(M_Int16 val)

Description

Converts network byte ordering of M_Int32 type value to host byte ordering

Return Value

Integer value

Side Effects

None

Reference Item

None

- **MC_utillnetAddrInt**

Prototype

M_int32 MC_utillnetAddrInt(M_Byte* addr)

Description

Gets the IP value having an integer type from a string IP address;

the IP value that is returned is network byte ordering

Return Value

Pass

IP value of network byte order

Fail

-1

Side Effects

None

Reference Item

None

- **MC_utillnetAddrStr**

Prototype

void MC_utillnetAddrStr(M_Int32 ip, M_Byte* addr)

Description

Gets an IP string from the IP value having an integer type; the integer type value should be network byte ordering

Parameters

ip	IP address having an integer type
addr	Buffer where the IP string is stored

Side Effects

None

Reference Item

None

2.9. Generic I/O

2.9.1. Overview

- **Definitions of Terms**

IrDA

Refers to a device that is used for infrared communication

1Chip

Refers to an IC card that is used to store personal information, which is necessary for authentication in electronic approval using a terminal

UICC (Universal IC Card)

Refers to a card that is used to store and manage various terminal information in a WCDMA terminal including 1Chip function

- **Overview**

This provides an API that is used to handle generic I/O device mounted on a terminal. Through this API, all I/O devices that will be added in the future can be controlled without adding other APIs. In the application, a device can be opened using the name of the corresponding device. Thereafter, the device can be controlled by Device ID. The type of devices supported by a terminal can be obtained through MC_knlGetSystemProperty(). For command, IODEVICES is used.

- **Functions and List of Functions**

Function	List
Device open/close	MC_ioDevOpen
	MC_ioDevClose
Data communication	MC_ioDevRead
	MC_ioDevWrite
Device control	MC_ioDevControl
Callback registration	MC_ioDevSetOpenCB
	MC_ioDevSetReadCB
	MC_ioDevSetWriteCB

- **Supported I/O Devices and Functions**

Currently specified and supported I/O devices include IrDA, Camera, and 1-chip device.

The following are the APIs and their functions for the currently supported devices:

IrDA

I/O Device Name	API	Function
IrDA	MC_ioDevOpen	Initializing the IrDA device
	MC_ioDevSetOpenCB	Registering a callback to get information on normal device connection, timeout, connection closing, etc.
	MC_ioDevClose	Closing the IrDA device
	MC_ioDevRead	Reading data through IrDA
	MC_ioDevSetReadCB	Registering a callback related to data reception
	MC_ioDevWrite	Writing data through IrDA
	MC_ioDevSetWriteCB	Registering a callback related to data transmission
	MC_ioDevControl	SETOPCODE - Setting the IrDA transmission method

1-Chip

I/O Device Name	API	Function
1ChipCard	MC_ioDevOpen	Initializing the 1Chip device
	MC_ioDevSetOpenCB	Registering a callback to get information on error status that can occur during 1Chip Card communication
	MC_ioDevClose	Closing the 1Chip device
	MC_ioDevRead	Reading data through the 1Chip device
	MC_ioDevSetReadCB	Registering a callback when receiving data
	MC_ioDevWrite	Writing data through the 1Chip device
	MC_ioDevSetWriteCB	Registering a callback when transmitting data
	MC_ioDevControl	GETSTATUS - Checking the existence of the IC Card
		GETCHANNEL - Getting a logical channel number

- **Control Information of MC_ioDevControl() by Command**

IrDA

Command	Parameter
SETOPCODE: Setting the IrDA transmission method	param1 [in] SETMETHOD string param2 [in] OBEXPUT string or OBEXGET string

1-Chip

Command	Parameter
GETSTATUS: Checking the IC Card insertion	param1 [in] Buffer pointer [out] "exist" string or "noexist" string param2 [in] Buffer size
GETCHANNEL: Getting a logical channel number of the currently allocated UICC	param1 [in] Integer-type pointer [out] Channel number param2 [in] Null

● DEVOPENCB

Prototype

typedef void(*DEVOPENCB)(M_Int32 fd, M_Int32 error, void *param)

Description

Callback function that is registered in the MC_ioDevSetOpenCB function; this function is called when an event related to normal device connection or abnormal closing occurs when the device is operating as a non-blocking device

Parameters

fd	Device ID
error	Pass 0
	Fail M_E_DEVCLOSE (when a close event occurs), M_E_TIMEOUT (when a timeout occurs), M_E_ERROR (in case of connection failure)
param	Callback parameter that is set when registering a callback function

Return Value

None

Side Effects

None

Reference Item

MC_ioDevSetOpenCB

● DEVREADCB

Prototype

typedef void(*DEVREADCB)(M_Int32 fd, M_Int32 error, void *param)

Description

Callback function that is registered in the MC_ioDevSetReadCB function; this function is called when the device can read data when operating as a non-blocking device

Parameters

fd Device ID

error Pass 0

 Fail M_E_ERROR

param Callback parameter that is set when registering a callback function

Return Value

None

Side Effects

None

Reference Item

MC_ioDevSetReadCB

- **DEVWRITECB**

Prototype

typedef void(*DEVWRITECB)(M_Int32 fd, M_Int32 error, void *param)

Description

Callback function that is registered in the MC_ioDevSetWriteCB function; this function is called when the device can transmit data during its operation as a non-blocking device

Parameters

fd Device ID
error Pass: 0; Fail: M_E_ERROR
param Callback parameter that is set when registering a callback function

Return Value

None

Side Effects

None

Reference Item

MC_ioDevSetWriteCB

● MC_ioDevOpen

Prototype

M_Int32 MC_ioDevOpen(M_Char *devname, M_Uint16 devnum, void *param)

Description

This function opens and initializes a device.

The name and number of supported devices can be obtained by transmitting IODEVICES as a parameter of the MC_knlGetSystemProperty() function. In case of more than two I/O devices that should be physically separated, they should be distinguished by devnum transmitted as a parameter. For example, when there are two IrDA devices, the first device becomes "0," and the second device, "1."

Parameters

[in]	devname	Name of the device
[in]	devnum	Number of the device
[in]	param	Parameter to be passed when opening the device

Return Value

Pass	Device ID	
Fail	M_E_NOTSUP	Issued when the device is unsupported
	M_E_INPROGRESS	Issued when the device is trying to make connection
	M_E_ISCONN	Issued when the device is already open
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

MC_knlGetSystemProperty

● MC_ioDevSetOpenCB

Prototype

*M_Int32 MC_ioDevSetOpenCB(M_Int32 fd, DEVOPENCB cb, void *param)*

Description

This function registers a callback function that is used to receive an event related to normal device connection or abnormal closing in case of a non-blocking I/O device. When the callback function is called, param is transmitted as a callback function API. On the other hand, when this function returns an error, the callback function is not called.

Parameters

[in]	fd	I/O device ID
[in]	cb	Callback function
[in]	param	Value transmitted when the callback function is called

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevClose

Prototype

M_Int32 MC_ioDevClose(M_Int32 fd)

Description

Closes the use of a device; in the case of a non-blocking I/O device, all callback functions registered in the device are deleted and will not be called

Parameters

[in] fd Device ID

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid

Side Effects

None

Reference Item

None

● **MC_ioDevRead**

Prototype

*M_Int32 MC_ioDevRead(M_Int32 fd, M_Byte *buf, M_Int32 len)*

Description

Reads data from an I/O device

Parameters

[in]	fd	Device ID
[out]	buf	Buffer pointer where the data to be read will be stored
[in]	len	Size of the buffer where the data to be read will be stored

Return Value

Pass	Length of data read
Fail	
	M_E_WOULDBLOCK Issued in the absence of data to be read
	M_E_BADFD Issued when the ID is invalid
	M_E_INVALID Issued when the buffer or buffer length is invalid

Side Effects

None

Reference Item

None

● MC_ioDevWrite**Prototype**

M_Int32 MC_ioDevWrite(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

Writes data in an I/O device

Parameters

[in]	fd	Device ID
[in]	buf	Buffer pointer where data to be written is stored
[in]	len	Size of the buffer where data to be written is stored

Return Value

Pass

Length of data written

Fail

M_E_WOULDBLOCK

Issued when the current data cannot be written

M_E_BADFD

Issued when the ID is invalid

M_E_INVALID

Issued when the buffer or buffer length is invalid

Side Effects

None

Reference Item

None

● **MC_ioDevSetReadCB**

Prototype

*M_Int32 MC_ioDevSetReadCB(M_Int32 fd, DEVREADCB cb, void *param)*

Description

This function registers a callback function that is called when data can be read through the device in case of a non-blocking I/O device. When the callback function is called, param is transmitted as an API of the callback function. In case the callback function to be registered is null, the callback function that was previously registered with this function is deleted. On the other hand, in case the callback function to be registered is null, the callback registered with this function is deleted. When this function returns an error, callback is not called. In contrast, when data can be read from the device after this function is called, the callback function is called.

Parameters

[in]	fd	I/O device ID
[in]	cb	Callback function
[in]	param	Value transmitted when the callback function is called

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevSetWriteCB

Prototype

*M_Int32 MC_ioDevSetWriteCB(M_Int32 fd, DEVWRITECB cb, void *param)*

Description

This function registers a callback function that is called when data can be read through the device in case of a non-blocking I/O device. When the callback function is called, param is transmitted as an API of the callback function. In case the callback function to be registered is null, the callback function that was previously registered with this function is deleted. On the other hand, in case the callback function to be registered is null, the callback registered with this function is deleted. When this function returns an error, the callback function is not called. In contrast, when the device can transmit data after this function is called, the callback function is called.

Parameters

[in]	fd	I/O device ID
[in]	cb	Callback function
[in]	param	Value transmitted when the callback function is called

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevControl

Prototype

```
M_Int32 MC_ioDevControl(M_Int32 fd, M_Char *cmd, void *param1, void *param2)
```

Description

Implements the relevant operation according to the command given to the device.

Parameters

[in]	fd	Device ID
[in]	cmd	String representing the type of operation to be implemented on the device
[in/out]	param1	Parameter to be passed to the relevant operation of the device or buffer pointer where the result value of the operation is stored
[in/out]	param2	Parameter to be passed to the relevant operation of the device or buffer pointer where the result value of the operation is stored

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the command is invalid
	M_E_ERROR	Issued when the implementation of the command failed

Side Effects

None

Reference Item

None

A IrDA Control

● MC_ioDevOpen

Prototype

*M_Int32 MC_ioDevOpen(M_Char *devname, M_Uint16 devnum, void *param)*

Description

This function initializes the IrDA device.

Specifically, this function sets whether the IrDA device will operate in server mode or client mode. In case of server mode, the IrDA device waits for a response request from the client for a specified period of time. On the other hand, in case of client mode, the IrDA device searches the server for a specified period of time; when the server exists, it attempts to connect to the server.

Parameters

devname	Use "IrDA."
devnum	The first device is given number 0, and other devices, subsequent numbers in sequence.
param	The mode to be set is transmitted using the following MC_IrDAMode structure. In case of server mode, "Server" is put in aszMode; in case of client mode, "Client" is put in aszMode. nTime specifies the time during which connection will be attempted.

```
typedef struct MC_IrDAMode
```

```
{
    M_Char aszMode[MC_IRDA_MODE_LEN];    //Mode specification
    M_Uint32 nTime;                       //Time during which
                                         connection will be attempted
} MC_IrDAMode;
```

Return Value

Pass	Device ID	
Fail		
	M_E_NOTSUP	Issued when IrDA is unsupported
	M_E_INPROGRESS	Issued when the device is trying to make connection
	M_E_ISCONN	Issued when the device is already connected
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevSetOpenCB

Prototype

M_Int32 MC_ioDevSetOpenCB(M_Int32 fd, DEVOPENCB cb, void *param)

Description

Receiving an event like the status of normal connection, timeout, and connection closing that occurs in the IrDA device requires registering a callback function using MC_ioDevSetOpenCB() API. 0 is transmitted as error value as a parameter of the callback function when normal connection was made, M_E_TIMEOUT, when the specified time lapsed, and M_E_DEVCLOSE, when the connection was closed. When IrDA communication is interrupted due to the circumstances of OEM, e.g., when there is an incoming call while IrDA communication is going on, M_E_OEMERROR is transmitted. In other error situations, M_E_ERROR is transmitted. When M_E_DEVCLOSE is received, IrDA device should be closed by calling MC_ioDevClose().

Parameters

fd	I/O device ID
cb	Callback function
param	Value transmitted when the callback function is called

Return Value

Pass

0

Fail

M_E_BADFD	Issued when the ID is invalid
M_E_INVALID	Issued when the parameter is invalid
M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● **MC_ioDevClose**

Prototype

M_Int32 MC_ioDevClose(M_Int32 fd)

Description

Closes the use of the IrDA device

Parameters

fd I/O device ID

Return Value

Pass	Device ID	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevRead

Prototype

M_Int32 MC_ioDevRead(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

This function receives data in accordance with the set mode through the IrDA device. First, the mode should be set in advance. Otherwise, M_E_ERROR is returned. When data cannot be received immediately, M_E_WOULDBLOCK is returned. In such cases, a callback function should be registered in MC_ioDevSetReadCB(); when it is possible to receive data, another attempt should be made to receive data within the callback function.

Parameters

fd	Device ID
buf	Buffer pointer where data to be read will be stored
len	Size of the buffer where data to be read will be stored

Return Value

Pass	Size of data read
Fail	
M_E_BADFD	Issued when the ID is invalid
M_E_INVALID	Issued when the buffer or buffer length is invalid
M_E_WOULDBLOCK	Issued when data cannot be read or written
M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevWrite

Prototype

*M_Int32 MC_ioDevWrite(M_Int32 fd, M_Byte *buf, M_Int32 len)*

Description

This function transmits data in accordance with the set mode through the IrDA device. First, the mode should be set in advance. Otherwise, M_E_ERROR is returned. When data cannot be transmitted immediately, M_E_WOULDBLOCK is returned. In such cases, a callback function should be registered in MC_ioDevSetWriteCB(); when it is possible to transmit data, another attempt should be made to transmit data within the callback function.

Parameters

fd	Device ID
buf	Buffer pointer where data to be read will be stored
len	Size of the buffer where data to be read will be stored

Return Value

Pass	Size of data written
Fail	
M_E_BADFD	Issued when the ID is invalid
M_E_INVALID	Issued when the buffer or buffer length is invalid
M_E_WOULDBLOCK	Issued when data cannot be read or written
M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevControl

Prototype

M_Int32 MC_ioDevControl(M_Int32 fd, M_Char *cmd, void *param1, void *param2)

Description

This function sets the OBEX transmission method through the IrDA device. Methods that can be set include OBEX PUT(0x82) and OBEX GET(0x83). The OBEX header value set through SETOPCODE maintains the transmission method unless it is reset again using MC_ioDevControl() API. When write is implemented immediately without setting the transmission method separately after opening the IrDA device, OBEX GET will be set as the default transmission method to configure the OBEX header.

Parameters

fd	I/O device ID
cmd	SETOPCODE used
param1	SETMETHOD used
param2	OBEXPUT used for OBEX PUT method; for OBEX GET method, OBEXGET is used

Return Value

Pass	0	
Fail	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the command is invalid
	M_E_ERROR	Issued when the implementation of the command failed

Side Effects

None

Reference Item

None

B 1Chip Control

● MC_ioDevOpen

Prototype

```
M_Int32 MC_ioDevOpen(M_Char *devname, M_Uint16 devnum, void *param)
```

Description

Activates the 1Chip Card; this function transmits the buffer and buffer length that are used to receive an ATR (Answer To Reset) for device reset to param

Parameters

devname	1ChipCard used for device name
devnum	The first device is given number 0, and other devices are given subsequent numbers in sequence.
param	The parameter that is transmitted when activating the IC Card is transmitted using MC_CardOption. The application allocates a buffer to bATR and passes the buffer size through wATRLen. The response value and its length are returned to bATR and wATRLen. When the size of the buffer passed is smaller than the actual response value, 0xFFFF and M_E_ERROR are returned to wATRLen.

```
typedef struct MC_CardOption
```

```
{
    M_Byte *bATR;           // ATR data pointer
    M_Uint16 wATRLen;       // ATR data length
} MC_CardOption
```

Return Value

Pass	Device ID
Fail	
	M_E_NODEVICE Issued when the card is not inserted
	M_E_BADFORMAT Issued when the transmission data format is invalid
	M_E_INVALID Issued when the parameter is invalid
	M_E_ERROR Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevClose**Prototype*****M_Int32 MC_ioDevClose(M_Int32 fd)*****Description**

Deactivates the 1Chip Card

Parameters

fd I/O device ID

Return Value

Pass

Device ID

Fail

M_E_NODEVICE

Issued when the card is not inserted

M_E_BADFD

Issued when the ID is invalid

M_E_ERROR

Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevRead

Prototype

M_Int32 MC_ioDevRead(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

Receives data through the card

Parameters

fd	Device ID
buf	Buffer pointer where data to be read will be stored
len	Size of the buffer where data to be read will be stored

Return Value

Pass

Size of data read

Fail

M_E_NODEVICE	Issued when the card is not inserted
M_E_BADFD	Issued when the ID is invalid
M_E_NOTACTIVE	Issued when the card is not active
M_E_INVALID	Issued when the buffer or buffer length is invalid
M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevWrite

Prototype

M_Int32 MC_ioDevWrite(M_Int32 fd, M_Byte *buf, M_Int32 len)

Description

Transmits data through the card

Parameters

fd	Device ID
buf	Buffer pointer where data to be read will be stored
len	Size of the buffer where data to be read will be stored

Return Value

Pass

Size of data written

Fail

M_E_BADFD	Issued when the ID is invalid
M_E_NODEVICE	Issued when the card is not inserted
M_E_BADFORMAT	Issued when the transmission data format is invalid
M_E_NOTACTIVE	Issued when the card is not active
M_E_INVALID	Issued when the buffer or buffer length is invalid
M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevControl

Prototype

M_Int32 MC_ioDevControl(M_Int32 fd, M_Char *cmd, void *param1, void *param2)

Description

Checks whether the IC Card is inserted

Parameters

fd	I/O device ID
cmd	GETSTATUS used for the command
param1	Allocates and transmits the buffer where the string is stored; return value is either "exist" or "noexist."
param2	Transmits the buffer size in integer value

Return Value

Pass	0	
Fail		
	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the parameter is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_ioDevControl**Prototype**

M_Int32 MC_ioDevControl(M_Int32 fd, M_Char *cmd, void *param1, void *param2)

Description

Used only for the WCDMA terminal; this function gets the logical channel number of the currently allocated UICC

Parameters

fd	I/O device ID
cmd	GETCHANNEL used for the command
param1	Transmits an integer-type pointer; returns the logical channel number of the currently allocated UICC
param2	Null

Return Value

Pass	0	
Fail		
	M_E_BADFD	Issued when the ID is invalid
	M_E_INVALID	Issued when the parameter is invalid
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

2.10. Terminal Resource

2.10.1. Overview

- **Definitions of Terms**

Terminal Resource

A general designation for data that are stored in the terminal area having specific data formats such as image, sound, address book, etc.

Terminal Resource Groups

Groups of resources that are classified according to the in-house service names (picture mate, music bell, photo, recording) of mobile communication service providers

Terminal Resource Function

Function that can access and use terminal resource groups and terminal resource

Resource Group Name

A name representing a resource group

Resource Name

A resource name denotes a unique ID that is used to differentiate resources in a resource group. A resource name is created in OEM and cannot be duplicated in an identical resource group.

Resource UI (User Interface) Name

A resource name that is displayed on the UI screen for a resource; this name can be created in a program and can be duplicated in an identical resource group.

Resource Group List

This is a list of resource groups that distinguishes each resource group name with “\0” character. There are two consecutive “\0” characters at the end. This is the format used as output value for terminal resource function.

Resource List

This is a list of resource names that distinguishes each resource name with “\0” character. There are two consecutive “\0” characters. This is the format used as output value for terminal resource function.

Phonebook/Private

Address book for a terminal that serves as a resource for a Phonebook/Private resource group

Phonebook/Group

Address book group for a terminal that serves as a resource for a Phonebook/Group resource group

Shortkey

Simpler than a telephone number, this shortkey is allocated to a specific telephone number of Phonebook/Private to serve as a resource for a Phonebook/Shortkey resource group. Through this shortkey, making a call is much easier.

Group Lock

This is a state wherein a resource group is locked. Before executing an API accessing a resource group in group lock, it is necessary to check the required password. A group lock is set or released regardless of the private lock.

Private Lock

This is a state wherein a resource is locked. Before executing an API that accesses a resource in the private lock, it is necessary to check the required password. A private lock is set or released regardless of the group lock.

Lock Setting

Setting the group lock on a resource group or the private lock on a resource; when setting the lock, it is necessary to check the required password

Lock Release

Releasing the group lock of a resource group or the private lock of a resource; when releasing the lock, it is necessary to check the required password

Lock Status

In case of a resource group with group lock support, this is the status indicating whether or not a group lock is set. In case of a resource of a resource group with private lock support, however, this is the status indicating whether or not private lock is set on a resource.

Lock Status and Resource Access

When accessing a resource like reading, writing, or deleting data of a resource, it is necessary to check the required password if the resource group that includes the resource is group-locked or the resource is private-locked.

Password

The password that is specified on a terminal

Image Resource

Resource that is shown on the display like an image or photo (including a resource that

shows several images like animated bitmap and video without sound)

Sound Resource

Resource that plays a bell or music sound

Resource that includes both image and sound

Resource that shows an image on the display and plays a sound

Specific Terminal State

Six terminal states, i.e., idle, incoming, poweron, poweroff, browseron, and browseroff

Unique ID

Unique ID for a resource that is allocated by the network provider or the contents provider

Group Information

This is the information on a specific resource group that can be obtained through a terminal resource function. Group information varies depending on the resource group and the type of group information.

Group Information Type

This is the type of group information. Group information of the specified resource group can be obtained according to the type of group information specified by the terminal resource function.

Resource Information

This is the information of a specific resource that can be obtained through the terminal resource function. Resource information varies depending on the resource and the type of resource information.

Resource Information Type

This is the type of resource information. Resource information of the specified resource can be obtained according to the type of resource information specified by the terminal resource function.

Search Word

This is the query that is used when searching resource through the terminal resource function. When searching resources, the terminal resource function returns a list of resources with search strings based on the search word type.

Search Word Type

This is the type of search word that is used when searching resources using the terminal resource function. When searching for resources, a different type of search is carried out depending on the search word type. Resource is searched using search

word type and search word.

Search Mode

Checks whether the search exactly matches a given string or includes a given string in accordance with the search mode

Write Mode

Determines whether a new resource should be created, or the existing resource should be overwritten when writing on the terminal resource

Resource Name Verification

Verifies whether or not the resource of a resource name specified in the specified resource group exists

wCard

wCard denotes the WIPI address book format that expanded and defined the necessary information from the terminal address book based on vCard 3.0, the standard name card data format.

● Overview

Terminal resource (hereinafter referred to as “resource”) is a general designation for data stored in the terminal area having specific data formats such as image, sound, address book, etc.

Groups of resources that are classified according to the in-house service names (picture mate, music bell, photo, recording) of mobile communication service providers are called terminal resource groups. All resources belong to each resource group.

Terminal resource functions provide channels through which resource groups and resources can be accessed by WIPI applications.

● Functions and List of Functions

Function	List
Management of terminal resources	MC_termResGetFormat
	MC_termResGetSize
	MC_termResGetUIName
	MC_termResExists
	MC_termResRead
	MC_termResWrite
	MC_termResDelete
	MC_termResRegister
	MC_termResGetRegisteredInfo
	MC_termResGetInfo
Management of terminal	MC_termResSearch
	MC_termResGetSupportedGroups

resource groups	MC_termResGetCount
	MC_termResGetList
	MC_termResGetGroupInfo
Terminal resource security	MC_termResGetGroupLockState
	MC_termResGetLockState
	MC_termResSetGroupLockState
	MC_termResSetLockState
	MC_termResCheckPassword
Terminal resource, etc.	MC_termResGetFreeSpace
	MC_termResExecuteCmd

● Terminal Resource Group

Each resource group consists of multiple resources, and each resource can have a specific data format. The following are pre-defined resource groups:

PICTUREMATE – Picture mate

MUSICBELL – Bell sound

PHOTO – Photo

VOICE – Voice recording

PHONEBOOK/PRIVATE – Address book (private)

PHONEBOOK/GROUP – Address book (group)

PHONEBOOK/SHORTKEY – Shortcut number

BLACKLIST – Blacklist; status of sound, vibration, and lamp set on a telephone number

SMSDATA/SENT – SMS data sent

SMSDATA/RECV – SMS data received

● List Format

When a terminal resource function uses a parameter for multiple data inputs or outputs, the separation of each data is effected by “\0”(null) character, and there are two consecutive “\0” characters at the end.

“PHOTO\0MUSICBELL\0\0”

“mypicture01\0mypicture02\0...mypicture100\0\0”

“MUSICBELL; mymusicbell\0PICTUREMATE; mypicture\0\0”

“0114441234\001199995555\0Friend\0Family\0\0”

“1\0100\0\0”

● Terminal Resource Name Format

Access to terminal resources is carried out through resource names. Accordingly, OEM provides resource names in a manner that avoids duplication, such that an application can use the resource name as a unique ID within the resource group. An application cannot specify a resource name when it creates a new resource; instead, it receives a

resource name provided by OEM.

- **Resource Name Format**

There is no restriction except that a resource should be unique within the resource group. The resource name is provided by the terminal.

- **Fixed Resource Name**

The following resource name is a fixed one (to access the corresponding data of a corresponding group, it is necessary to use a given resource name):

Resource Group	Resource Name	Remarks
Phonebook/Shortkey	Specified shortkey string	Ex.) "5," "49"

- **Resource UI (User Interface) Name**

A resource name displayed on the UI screen by the resource; depending on the resource group, there can be a resource group without a UI name (in case of PHONEBOOK/GROUP and PHONEBOOK/SHORTKEY groups, however, the resource name and the UI name should be the same).

- **Resource Name and UI Name**

A resource has a resource name and a UI name. A resource name is a unique ID that distinguishes resources within a resource group. The resource name for the Phonebook/Shortkey group is specified in the Specification, and resource names for other resources are managed by the terminal. The application receives a list of resource names provided by the terminal for use. When creating a new resource, the application receives a resource name from the terminal. An index or other forms of string can become a resource name. Since the terminal provides a unique name that is not duplicated, however, only the application can access the resource.

On the other hand, the UI name is a name that is shown to the user through UI.

For example, in the case of a photo group, the name set directly by the user after he/she obtained an image from camera shooting is a UI name. When the application shows to the user an image list of the photo groups stored in the terminal, the UI name is shown on the terminal display. Even when the terminal creates a photo name by itself instead of the user, if the application has to show a photo list, a list of photo names should be shown. The UI name is required in such cases. As another example, if it is a bell sound of a Musicbell group, there will be names that are sent to the user such as "Star in my heart," "A Maiden of Soyang River," and "Return to the Busan Harbor" for each resource. This kind of name sent to the user is the UI name.

Depending on the resource, however, a resource may not have a UI name. The case of

SMS data belongs to this category. With SMS data, the UI name is not necessary since a message list commonly shows the first portion of a message to the user. In other words, the resource name is an ID that is used to utilize a resource, whereas the UI name is the name that is shown to the user through UI. A resource name can be duplicated in a resource group. While the resource name is given to all resources, the UI name is given only to the resource of the resource group that requires the UI name. While the resource name cannot be set or changed by an application, the UI name can be set or changed by an application.

● MIME Type and Resource Data Format

Each terminal resource has the following MIME types and data formats:

Image Format

MIME Type	Data Format
image/bmp	Bitmap image format
image/gif	GIF image format
image/jpeg	JPEG image format
image/png	PNG image format
image/sis	SIS image format

Animation Format

MIME Type	Data Format
anim/sis	SIS image format
anim/gif	GIF image format

Video Format

MIME Type	Data Format
video/MPEG4	Mpeg4
video/H.263	H.263
video/H.264	H.264

Sound Format

MIME Type	Data Format
Qualcomm_CMX	Qualcomm CMX
Yamaha_MA1	Yamaha MA1
Yamaha_MA2	Yamaha MA2
Yamaha_MA3	Yamaha MA3 single channel format
Yamaha_MA5	Yamaha MA3
Yamaha_SMAF	Yamaha single channel format
Yamaha_SMAF-Phrase	Yamaha multi-channel format
Yamaha_SMAF-Audio	Yamaha SMAF-audio format
audio/ONEPOLY	One Poly Media Format
audio/GVMONEPOLY	GVM One Poly Media Format

audio/MIDI	MIDI
audio/MP3	MP3
audio/TONE	Tone
audio/FREQTONE	Frequency tone
IS96	QCELP-8K
IS96A	QCELP-8K
IS733	QCELP-13K
IS127	EVRC-8K
G.723.1	G.723.1
audio/AAC	AAC
audio/AAC+	AAC+
AMR-WB	Voice codec for WCDMA
AMR-NB	

Phonebook Private, Phonebook Group, and Shortcut Format

MIME Type	Data Format
phonebook/private	wCard string
phonebook/group	Actual group name string: Name shown through UI
phonebook/shortcut	<p><Resource name of phonebook private> + "/" + <Type and value of wCard TEL type></p> <p>Ex.) When the home telephone number of a phonebook private resource whose resource name is "Kim Cheol-su" is 02-1234-5678,</p> <p>"Kim Cheol-su/TEL; TYPE=home:0212345678"</p>

BLACKLIST Format

MIME Type	Data Format
blacklist	<p>blacklist data = NUMBER + ";" + STATE</p> <p>Number: Incoming call number (String format without "-")</p> <p>State: blacklist setting status</p> <p>SND: When the setting is sound status</p> <p>VIB: When the setting is vibration status</p> <p>LMP: When the setting is lamp status</p> <p>Ex.) 011-1234-5678 is set as lamp status,</p> <p>"01112345678; LMP"</p>

SMS Incoming/Outgoing Data Format

MIME Type	Data Format
smsdata	<p>SMS data = Index + "\0" + State + "\0" + Number + "\0" + Data + "\0" + Time + "\0" + "\0"</p> <p>Index: Internal management number string of SMS message</p> <p>State: State of the message</p> <p>0: New message</p> <p>1: Message already read</p> <p>Number: Telephone number string of the sender or the receiver ("-" None)</p> <p>Data: SMS message string with ASCII format</p> <p>Time: Transmission/reception time string (Year Month Day Hour Minute (yyyymmddhhmm) format)</p> <p>Ex.) When a "Hello" message has arrived from phone number 011-1234-5678 at 12:55, May 4, 2003:</p> <p>"10\00\001112345678\0Hello\0200305041255\0\0"</p>

- **Terminal Resource Management Function**

The terminal resource management function provides a function that can get information on data format and data size of a specific resource, a function that reads or writes data in a resource, and a function that deletes a resource. In addition, it can set a specific resource for specific terminal states (i.e., POWERON, POWEROFF, IDLE, INCOMING, BROWSEON, and BROWSEOFF) and get the set information. It also gets information on a specific resource, searches resources, and verifies whether there are resources with specific names.

- **Terminal Resource Group Management Function**

This function gets resource groups that are supported by the terminal. It also gets information on the number of resources of a specific resource group, a list of resource names, and other information on the resource group.

- **Terminal Resource Security Function**

This function can get the lock status on a specific resource group and resources and set or release relevant locks. It also receives a password from the user and compares it with the password registered in the terminal to determine whether the two passwords match.

When accessing a resource like reading or writing data in a group-locked resource of a resource group or reading or writing data in a private-locked resource, the application should verify whether the password received from the user matches the password

registered in the terminal. When setting or releasing locks, the application should also check the password.

● Other Functions of the Terminal Resource

This function gets the size of the available space of terminal resources. It can also implement specific services in accordance with a specified command. It gets the size of the available space of the terminal resource storage space, and it can also implement a specific service in accordance with a specified command.

● Available API List by Resource Group

The following represents the available resource groups for APIs specifying resource groups among terminal resource APIs (APIs that do not specify resource groups are not related to this table):

Resource Group	Available API
All resource groups	MC_termResGetCount MC_termResGetList MC_termResGetFormat MC_termResGetSize MC_termResExists MC_termResRead
By terminal	MC_termResGetGroupLockState MC_termResSetGroupLockState MC_termResGetLockState MC_termResSetLockState MC_termResGetUIName
By parameter	MC_termResGetGroupInfo (vary according to infoType) MC_termResGetInfo (vary according to info Type) MC_termResSearch (vary according to queryType) MC_termResExecuteCmd (vary according to command)
PICTUREMATE MUSICBELL PHOTO VOICE	MC_termResWrite MC_termResDelete MC_termResRegister
PHONEBOOK/PRIVATE PHONEBOOK/SHORTKEY PHONEBOOK/GROUP BLACKLIST SMSDATA/SENT	MC_termResWrite MC_termResDelete
SMSDATA/RECV	MC_termResDelete

- **Lock State**

Prototype

```
#define MC_TERMRES_GROUP_LOCK  
#define MC_TERMRES_GROUP_UNLOCK  
#define MC_TERMRES_PRIVATE_LOCK  
#define MC_TERMRES_PRIVATE_UNLOCK
```

Description

Represents the lock state of a terminal resource group or a terminal resource

Reference Item

- MC_termResGetGroupLockState
- MC_termResGetLockState
- MC_termResSetGroupLockState
- MC_termResSetLockState

- **Search Mode**

Prototype***#define MC_TERMRES_EXTSRCH******#define MC_TERMRES_INCSRCH*****Description**

Determines the search mode when searching a terminal resource

Reference Item

MC_termResSearch

- **Write Mode**

Prototype***#define MC_TERMRES_CREATE******#define MC_TERMRES_UPDATE*****Description**

Determines the write mode when writing in a terminal resource

Reference Item

MC_termResWrite

- **Resource Name Verification**

Prototype***#define MC_TERMRES_NAME_EXISTENT******#define MC_TERMRES_NAME_NONEXISTENT*****Description**

Represents whether the resource of a corresponding resource name exists in the resource group

Reference Item

MC_termResExists

● MC_termResGetSupportedGroups

Prototype

M_Int32 MC_termResGetSupportedGroups(M_Byte resGroup, M_Int32 bufSize)*

Description

Returns a list of resource groups that are supported by the terminal

Parameters

[out]	resGroup	Resource group list (in the list format of the terminal resource overview)
[in]	bufSize	resGroup buffer size

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_SHORTBUF	Issued when the size of a buffer where the return value is contained is too small

Side Effects

None

Reference Item

None

● MC_termResGetCount**Prototype*****M_Int32 MC_termResGetCount(M_Char* resGroupName)*****Description**

Returns the number of resources belonging to a specified resource group

Parameters

[in] resGroupName Resource group name

Return Value

Pass

Number of resources

Fail

M_E_ERROR

Issued in case of failure due to unknown reasons

M_E_INVALID

Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

None

● MC_termResGetList

Prototype

M_Int32 MC_termResGetList(M_Char* resGroupName, M_Byte* aszList, M_Int32 bufSize)

Description

Returns a list of resources belonging to a specified resource group

Parameters

[in]	resGroupName	Resource group name
[out]	aszList	Resource name list (in the list format of the terminal resource overview)
[in]	bufSize	aszList buffer size

Return Value

Pass

0

Fail

M_E_ERROR

Issued in case of failure due to unknown reasons

M_E_SHORTBUF

Issued when the transmitted buffer size is smaller than the string to be returned

M_E_INVALID

Issued when the transmitted parameter is Invalid

Side Effects

None

Reference Item

None

● **MC_termResGetFormat**

Prototype

M_Int32 MC_termResGetFormat(M_Char* resGroupName, M_Char* resName, M_Char* rtnFormat, M_Int32 rtnFormatSize)

Description

Returns a MIME type string of the specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[out]	rtnFormat	MIME type string
[in]	rtnFormatSize	rtnFormat buffer size

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_SHORTBUF	Issued when the transmitted buffer size is smaller than the string to be returned
	M_E_INVALID	Issued when the transmitted parameter is Invalid
	M_E_NOTSUPPORTTYPE	Resource without MIME type

Side Effects

None

Reference Item

None

● MC_termResGetSize**Prototype**

M_Int32 ***MC_termResGetSize(M_Char* resGroupName, M_Char* resName)***

Description

Returns the data size of a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass	Resource size
Fail	
	M_E_ERROR Issued in case of failure due to unknown reasons
	M_E_INVALID Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

Used to allocate the required buffer when calling MC_termResRead to read a resource

● **MC_termResGetUIName**

Prototype

M_Int32 MC_termResGetUIName(M_Char resGroupName, M_Char* resName, M_Char* uiName, M_Int32 uiNameSize)*

Description

Returns the UI name of a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[out]	uiName	Name that appears on UI
[in]	uiNameSize	uiName buffer size

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_SHORTBUF	Issued when the transmitted buffer size is smaller than the string to be returned
	M_E_INVALID	Issued when the transmitted parameter is invalid (also applicable to a resource group that does not support the UI name)

Side Effects

None

Reference Item

None

● MC_termResExists

Prototype

M_Int32 MC_termResExists(M_Char resGroupName, M_Char* resName)*

Description

Verifies whether the resource of a specified resource name exists

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass	MC_TERMRES_NAME_EXISTENT	
	The resource of a corresponding resource name exists in the resource group.	
	MC_TERMRES_NAME_NONEXISTENT	
	The resource of a corresponding resource name does not exist in the resource group.	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_INVALID	Issued when the transmitted parameter is invalid

Side Effects

None

● MC_termResRead

Prototype

M_Int32 MC_termResRead(M_Char resGroupName, M_Char* resName,
M_Byte* pData, M_Int32 bufSize)*

Description

Returns the data of a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[out]	pData	Resource data
[in]	bufSize	pData buffer size

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_SHORTBUF	Issued when the transmitted buffer size is smaller than the string to be returned
	M_E_INVALID	Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

The caller allocates as much pData as the size of the resource obtained through MC_termResGetSize().

● MC_termResWrite

Prototype

***M_Int32 MC_termResWrite(M_Char* resGroupName, M_Char* resName,
M_Int32 nameSize, M_Char* uiName, M_Char* resFormat, M_Byte* pData,
M_Int32 bufSize, M_Int32 mode)***

Description

Records or updates a resource to a specified resource group; in case of recording, a new resource name created is returned (when a UI name is not assigned, it can be randomly given by OEM)

Parameters

[in]	resGroupName	Resource group name
[in/out]	resName	New or already existing resource name
[in]	nameSize	resName buffer size
[in]	uiName	Name that appears on UI
[in]	resFormat	MIME type of the resource
[in]	pData	Resource data
[in]	bufSize	pData buffer size
[in]	mode	Write mode
	MC_TERMRES_CREATE	Creates a new resource; for resName, the new resource name created is returned
	MC_TERMRES_UPDATE	The existing resource is overwritten. For resName, the new resource name is given.

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown Reasons
	M_E_INSUFSPACE	Issued in case of insufficient resource storage space
	M_E_INVALID	Issued when the transmitted parameter is invalid
	M_E_ACCESSDENY	Issued when the user has no write authority for the resource
	M_E_NOTSUP	Issued when the write function is not

	supported for the resource group
M_E_INVALIDDATA	Issued when the data does not comply with the relevant data format

Side Effects

In the MC_TERMRES_CREATE mode, the UI name passed as an argument can be ignored if there is a specific regulation on the UI name in OEM, or the resource group has no UI name.

On the other hand, in the MC_TERMRES_UPDATE mode, the function operates if the resource group name, resource name, and information of the resource with MIME type all match, and the existing data is updated with a new one.

In case of a resource group supporting the UI name, the UI name should be specified. Otherwise, an error or M_E_INVALID is issued.

Reference Item

The resource of the Phonebook/Shortkey group uses the UI name as a shortcut. Therefore, the resource of the PHONEBOOK/SHORTKEY group should have a matching resource name with the UI name. Likewise, the UI name can be used to specify the shortcut.

● MC_termResDelete

Prototype

M_Int32 MC_termResDelete(M_Char* resGroupName, M_Char* resName)

Description

Deletes a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass

0

Fail

M_E_ERROR	Issued in case of failure due to unknown reasons
M_E_INVALID	Issued when the transmitted parameter is invalid
M_E_ACCESSDENY	Issued when the user has no access right for the resource
M_E_NOTSUP	Issued when the delete function is not supported by the resource group
M_E_NODELETE	Issued when the resource cannot be deleted

Side Effects

None

Reference Item

None

● MC_termResRegister

Prototype

M_Int32 MC_termResRegister(M_Char* resGroupName, M_Char* resName, M_Char* szStatus)

Description

Sets a specified resource to a specific terminal status

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[in]	szStatus	Specific terminal status
	IDLE	Idle
	INCOMING	Incoming call
	POWERON	Power on
	POWEROFF	Power off
	BROWSEON	Browser on
	BROWSEOFF	Browser off

Return Value

Pass

0

Fail

M_E_ERROR	Issued in case of failure due to unknown reasons
M_E_INVALID	Issued when the transmitted parameter is invalid
M_E_ACCESSDENY	Issued when the user has no setting authority
M_E_INVALIDSTATUS	Issued when the resource has no relation to a specific terminal status
M_E_NOTSUP	Issued when the register function is not supported by the resource group

Side Effects

None

Reference Item

None

● MC_termResGetRegisteredInfo

Prototype

M_Int32 MC_termResGetRegisteredInfo(M_Byte* resList, M_Int32 bufSize, M_Char* szStatus)

Description

Returns the resource group name and the resource name set to a specific terminal status

Parameters

[out] resList

The list of resource groups and resource names is made up of resource group name, semicolon (;), and resource name. In case of two resources, the distinction between a resource group name and a resource name follows the list format of the terminal resource overview.

Ex.) "MUSICBELL;mymusicbell\0\0"

"MUSICBELL;mymusicbell\0PICTUREMATE;mypicture\0\0"

[in] bufSize resList buffer size

[in] szStatus Specific terminal status

IDLE Idle screen

INCOMING Incoming call screen

POWERON Power on screen

POWEROFF Power off screen

BROWSERON Browser on

BROWSEROFF Browser off

Return Value

Pass

0

Fail

M_E_ERROR Issued in case of failure due to unknown reasons

M_E_INVALID Issued when the transmitted parameter is invalid

M_E_ACCESSDENY Issued when the user has no setting authority

M_E_INVALIDSTATUS Issued when the resource has no relation to a specific terminal status

M_E_NORES	Issued when there is no resource set to a specific terminal status
M_E_SHORTBUF	Issued when the resNames buffer size is too small
M_E_NOTSUP	Issued when the register function is not supported by the resource group

Side Effects

None

Reference Item

Up to two resources are set to a specific terminal status.

● MC_termResGetGroupLockState

Prototype

M_Int32 MC_termResGetGroupLockState (M_Char* resGroupName)

Description

Returns the lock state of a specified resource group

Parameters

[in] resGroupName Resource group name

Return Value

Pass

MC_TERMRES_GROUP_LOCK

A group lock is set on the resource group.

MC_TERMRES_GROUP_UNLOCK

A group lock is not set on the resource group.

Fail

M_E_ERROR

Issued in case of failure due to unknown reasons

M_E_NOTSUPPORTLOCK

Lock setting is not supported by the resource group or the resource.

M_E_NOTSUPPORTGLOCK

Group lock is not supported by the resource group (only private lock is supported).

M_E_INVALID

Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

None

● MC_termResGetLockState

Prototype

M_Int32 MC_termResGetLockState(M_Char* resGroupName, M_Char* resName)

Description

Returns the lock state of a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name

Return Value

Pass

MC_TERMRES_PRIVATE_LOCK

A private lock is set on the resource.

MC_TERMRES_PRIVATE_UNLOCK

A private lock is not set on the resource.

Fail

M_E_ERROR

Issued in case of failure due to unknown reasons

M_E_NOTSUPPORTLOCK

The lock setting is not supported by the resource group or the resource.

M_E_NOTSUPPORTPLOCK

Private lock is not supported by the resource group (only group lock is supported).

M_E_INVALID

Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

None

● MC_termResSetGroupLockState

Prototype

M_Int32 MC_termResSetGroupLockState(**M_Char*** resGroupName,
M_Int32 state)

Description

Sets the lock state of a specified resource group

Parameters

[in]	resGroupName	Resource group name	
[in]	state	Group lock setting/release	
		MC_TERMRES_GROUP_LOCK	Group lock setting
		MC_TERMRES_GROUP_UNLOCK	Group lock release

Return Value

Pass

0

Fail

M_E_ERROR

Issued in case of failure due to unknown reasons

M_E_NOTSUPPORTLOCK

The lock setting is not supported by the resource group or the resource.

M_E_NOTSUPPORTGLOCK

Group lock is not supported by the resource group (only private lock is supported).

M_E_INVALID

Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

Check the password before use through MC_termResCheckPassword.

● MC_termResSetLockState

Prototype

M_Int32 MC_termResSetLockState(M_Char resGroupName, M_Char* resName, M_Int32 state)*

Description

Sets the lock state of a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[in]	state	Private lock setting/release
	MC_TERMRES_PRIVATE_LOCK	Private lock setting
	MC_TERMRES_PRIVATE_UNLOCK	Private lock release

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_NOTSUPPORTLOCK	The lock setting is not supported by the resource group or the resource.
	M_E_NOTSUPPORTPLOCK	Private lock is not supported by the resource group (Only the group lock is supported).
	M_E_INVALID	Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

Check the password before use through MC_termResCheckPassword.

● MC_termResCheckPassword

Prototype

M_Int32 MC_termResCheckPassword(M_Char szPassword)*

Description

Checks whether the specified password matches the password registered in the terminal

Parameters

[in] szPassword Password

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_INCORRECTPASSWORD	Issued when the password does not match
	M_E_INVALID	Issued when the transmitted parameter is invalid

Side Effects

None

Reference Item

None

- **MC_termResGetFreeSpace**

Prototype

M_Int32 MC_termResGetFreeSpace(void)

Description

Returns the available storage space of a specified resource group

Parameters

None

Return Value

Pass

Available storage space of a specified resource group

Fail

M_E_INVALID

Issued when the transmitted parameter is invalid

M_E_ERROR

Issued in case of failure due to unknown reasons

M_E_NOTSUP

Issued when the resource group is not supported

● MC_termResGetGroupInfo

Prototype

M_Int32 MC_termResGetGroupInfo(M_Char* resGroupName, M_Char* infoType, M_Byte* infoData, M_Int32 bufSize)

Description

Returns the group information with a specified group information type for a specific resource group

Parameters

[in]	resGroupName	Resource group name
[in]	infoType	Group information type (see the definition in the Reference Item below)
[out]	infoData	Group information
[in]	bufSize	infoData buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Issued in case of failure due to unknown reasons
M_E_INVALID	Issued when the transmitted parameter is invalid
M_E_SHORTBUF	Issued when the group information buffer size is too small
M_E_NOTSUPPORTTYPE	Issued when the group does not support the group information of a specified group information type
M_E_NOENT	Issued in the absence of a group information with a specified group information type

Side Effects

Only the resource group corresponding to the infoType defined in the Reference Item below can be specified as resGroupName.

Reference Item

'+' : Refers to an operation that strcats the preceding or following string/character.

Resource Group	infoType	Remarks
All groups	MAXUINAMESIZE	Returns the maximum length of the UI name that can be stored in a specified resource group in a string

PHONEBOOK/PRIVATE PHONEBOOK/GROUP SMSDATA/SENT SMSDATA/RECV BLACKLIST	MAXCOUNT	Returns the maximum number of resources that can be stored in a specified resource group in a string
PHONEBOOK/SHORTKEY	RANGE	Returns the starting number and ending number of a shortcut supported by the terminal in a string; the string shall follow the 3.9.1 format in descending order from the starting number and the ending number Ex. 1) "1\0100\0\0" (for a terminal that can use a number between 1 and 100 as a shortcut) Ex. 2) "0\0200\0\0" (for a terminal that can use a number between 1 and 200 as a shortcut)
PHONEBOOK/GROUP	IRREMOVABLE	Returns a resource list that cannot be deleted; the list follows the "List Format" of the module porting of a terminal resource (in the absence of a resource that cannot be deleted, an error or M_E_NOENT is returned)
PHONEBOOK/PRIVATE	MAXTELCOUNT	Returns the maximum number of telephone numbers that can be stored in a terminal in a string
	TYPELIST	Returns a list of wCard types supported by the terminal; the list follows the "List Format" of the module porting of a terminal resource
	TYPECOUNT	Returns the number of wCard types supported by the terminal
	TYPEINFO + "/" + <Type name> Ex.) TYPEINFO/N TYPEINFO/TEL	Returns the wCard type information corresponding to the wCard type name; the return value shall be effected in string format as: <Length of a value that can be stored> + "/" + <Number of types that can be stored in Phonebook Private> [+ "/" + <Type parameter> + ":" <Number of type parameters that can be stored in Phonebook Private>] Examples of use are provided below:

	"X-MDAYINFO"	<p>Gets anniversary information from the terminal The following is the return format: <Anniversary> + '/' + <Fixed or Variable> (+ '/' + <Maximum length of VARIABLE>) (+ "/NOLEAF") <Anniversary>: YYYYMMDD or MMDD <Fixed or Variable>: FIXED or VARIABLE <Maximum length of Variable>: Maximum length of the anniversary's Type Parameter that can be entered using ASCII CHAR "/NOLEAF": In case of a terminal wherein the leap month cannot be entered in the lunar calendar Refer to wCard Ex. 1) In case the terminal supports YEAR in the anniversary, and the anniversary is FIXED; if the lunar calendar is supported, the leap month is supported as well "YYYYMMDD/FIXED" Ex. 2) In case the terminal supports YEAR in the anniversary, and the anniversary is FIXED; the lunar calendar is supported but not the leap month "YYYYMMDD/FIXED/NOLEAF" Ex. 3) In case the terminal does not support YEAR in the anniversary, and the user directly enters the type of anniversary (maximum available length: 6) "MMDD/VARIABLE/6" The terminal that does not support X-MDAY returns an error (M_E_NOENT).</p>
--	--------------	---

Example of "TYPEINFO"**Ex. 1) Returns "10/1" using TYPEINFO/N as infoType**

- The length of a value that can be stored in N type is 10 ASCII type characters.
- One N type can be stored in Phonebook Private.
- There is no type parameter in N type.

Ex. 2) Returns "15/4/cell:4/work:4/home:4" using TYPEINFO/TEL as infoType

- The length of a value that can be stored in TEL type is 15 ASCII type characters.
- Four TEL types can be stored in Phonebook Private.
- In TEL type, there are three type parameters of cell, work, and home.
- In one Phonebook Private, four type parameters of cell, work, and home can be stored.

Description: In one Phonebook Private, four telephone numbers can be stored including home and company telephone numbers and mobile phone number. All four numbers can be set to home telephone number or mobile phone number.

Ex. 3) Using TYPEINFO/X-MDAY as infoType

- "10/4/birthday:1/wedding:1/meeting:1/memorial:1/sun:4/moon:4" is returned.
- The length of a value that can be stored in X-MDAY type is 10 ASCII type

characters.

- Four X-MDAY types can be stored in Phonebook Private.
- In X-MDAY type, there are six type parameters of birthday, wedding, meeting, memorial, sun, and moon.
- One birthday, wedding, meeting, and memorial type parameter can be stored in one Phonebook Private.
- Four sun and moon type parameters can be stored in one Phonebook Private..

Description: In one Phonebook Private, four events or one birthday, wedding, meeting, and memorial each can be stored. Accordingly, four events including birthday, wedding, meeting, and memorial can be stored in one Phonebook Private. These events can be stored either in the Julian calendar or lunar calendar.

● MC_termResGetInfo

Prototype

M_Int32 MC_termResGetInfo(M_Char* resGroupName, M_Char* resName, M_Char* infoType, M_Byte* infoData, M_Int32 bufSize)

Description

Returns the resource information of a specified resource information type for a specified resource

Parameters

[in]	resGroupName	Resource group name
[in]	resName	Resource name
[in]	infoType	Resource information type (see definition in the Reference Item below)
[out]	infoData	Resource information
[in]	bufSize	infoData buffer size

Return Value

Pass

0

Fail

M_E_ERROR	Issued in case of failure due to unknown reasons
M_E_INVALID	Issued when the transmitted parameter is invalid
M_E_SHORTBUF	Issued when the resource information buffer size is too small
M_E_NOTSUPPORTTYPE	Issued when the resource does not support the resource information with a specified resource information type
M_E_NOENT	Issued in the absence of a resource information with a specified resource information type in the resource

Side Effects

Only the resource corresponding to the resource format defined in the Reference Item below can be specified.

Reference Item

“+”: Refers to an operation that strcats the preceding or following string/character.

The resource format follows the “resource data format and MIME type” described in the terminal resource overview.

Resource Format	infoType	Remarks
Image format Animation format Video format	WIDTH	When there is width information in the specified resource, this information is returned in a string. The unit is pixel.
	HEIGHT	When there is height information in the specified resource, this information is returned in a string. The unit is pixel.
Video format Sound format	RUNTIME	When there is running time information in the specified resource, this information is returned in a string. The unit is ms.
	BITRATE	When there is bit rate information in the specified resource, this information is returned in a string. The unit is bps (bit per sec).
Video format	FRAMERATE	When there is frame rate information in the specified resource, this information is returned in a string. The unit is fps (frame per sec).
Phonebook/Group format	PRIVATECOUNT	Returns the number of Phonebook Privates contained in a specified Phonebook Group in a string; in case the Phonebook Group does not have Phonebook Private, 0 is returned
	PRIVATELIST	Returns the resource name list of a Phonebook Private contained in a specified Phonebook Group; the list shall follow the “List Format” of the terminal resource overview (in case the Phonebook Group does not have Phonebook Private, an error or M_E_NOENT is returned)

● MC_termResSearch

Prototype

M_Int32 MC_termResSearch(M_Char* resGroupName, M_Char* queryType, M_Char* queryName, M_Byte* resNames, M_Int32 bufSize, M_Int32 mode)

Description

Searches the resource matching a given string search word depending on the resource group and search word type; a resource name list is returned as a result of the search

Parameters

[in]	resGroupName	Resource group name
[in]	queryType	Search word type (see definition in the Reference Item below)
[in]	queryName	String search word
[out]	resNames	Resource name list (follows the “list format” of the overview document)
[in]	bufSize	resNames buffer size
[in]	mode	Search mode
		MC_TERMRES_EXTSRCH
		Checks whether the resource exactly matches the queryName string
		MC_TERMRES_INCSRCH
		Checks whether the resource includes the queryName string

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_SHORTBUF	Issued when the transmitted buffer size is smaller than the string to be returned
	M_E_INVALID	Issued when the transmitted parameter is invalid
	M_E_NOTSUPPORTTYPE	Issued when the resource group does not support the queryType
	M_E_NOTFOUND	Issued when no resource was found in the search

Side Effects

Only the resource group corresponding to the queryType specified in the Reference Item below can be specified as resGroupName.

Reference Item

Resource Group	queryType	Remarks
All resource groups	UINAME	Searches the UI name resource with the queryName string in a specified resource group (a resource group that does not support the UI name can be included)
PHONEBOOK/PRIVATE	NUMBER	Telephone number search: Searches Phonebook Private with the queryName string in the telephone number

● MC_termResExecuteCmd

Prototype

M_Int32 MC_termResExecuteCmd(M_Char* resGroupName, M_Char* cmd, void* param1, void* param2)

Description

Requests the service based on a specified command

Parameters

[in]	resGroupName	Resource group name for which a command is executed
[in]	cmd	Command through which a service is received (see definition in the Reference Item below)
[in/out]	param1	Argument/return value for the service (see definition in the Reference Item below)
[in/out]	param2	Argument/return value for the service (see definition in the Reference Item below)

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of failure due to unknown reasons
	M_E_INVALID	Issued when the transmitted parameter is invalid
	M_E_NOTSUP	Issued when the command is not supported; to be defined for each command (see Reference Item below)

2.11. Media Handler

This is a package for functions, tone reproduction, recording, and volume adjustment related to a media handler handling all media including sound and video image.

All data such as sound, tone, and video image are abstracted into a clip and implemented in a media handler. The types supported by a media player are those that have been obtained by MEDIADVICES of MC_knlGetSystemProperty().

The status change of media handling, tone reproduction, recording, etc. are transmitted to the callback function to be registered. Volume adjustment is possible for tone, sound, and recording.

- **MC_MDA_STATUS_ERROR**

Prototype

#define MC_MDA_STATUS_ERROR (-1)

Description

Stopped due to an error; the constant value is -1

- **MC_MDA_STATUS_END_OF_DATA**

Prototype***#define MC_MDA_STATUS_END_OF_DATA 1*****Description**

The state when the media handler reached the end of media (or tone) during handling media (or tone); the constant value is 1

- **MC_MDA_STATUS_START**

Prototype***#define MC_MDA_STATUS_START 2*****Description**

The state when the media handler began handling media (or tone) during handling media (or tone); the constant value is 2

- **MC_MDA_STATUS_STOP**

Prototype***#define MC_MDA_STATUS_STOP 3*****Description**

When handling media (or tone), this is the state when the media handler completed the handling of media (or tone). This is also the state when recording is stopped during recording. The constant value is 3.

- **MC_MDA_STATUS_PAUSE**

Prototype***#define MC_MDA_STATUS_PAUSE 4*****Description**

When handling media (or tone), this is the state when the handling of media (or tone) is temporarily stopped. This is also the state when recording is temporarily stopped during recording. The constant value is 4.

- **MC_MDA_STATUS_RESUME**

Prototype***#define MC_MDA_STATUS_RESUME 5*****Description**

When handling media (or tone), this is the state when handling of media (or tone) is resumed after a pause. This is also the state when recording is resumed after a pause during recording. The constant value is 5.

- **MC_MDA_STATUS_RECORD**

Prototype

#define MC_MDA_STATUS_RECORD 6

Description

The state when recording started during recording; the constant value is 6

- **MC_MDA_STATUS_FULL_OF_DATA**

Prototype

```
#define MC_MDA_STATUS_FULL_OF_DATA 7
```

Description

The state when the internal buffer of a clip is completely filled during recording; the constant value is 7

- **MC_MDA_STATUS_OEM_ERROR**

Prototype***#define MC_MDA_STATUS_OEM_ERROR 8*****Description**

Issued when the media being played or voice recorded is terminated while the platform is sent to the background during the playing of media or recording; the constant value is 8

● Tone Note value

```
typedef enum _MC_MdaToneType {
    MC_MDA_TONE_0 = 0           //DTMF for 0 key
    MC_MDA_TONE_1 ,             //DTMF for 1 key
    MC_MDA_TONE_2 ,             //DTMF for 2 key
    MC_MDA_TONE_3 ,             //DTMF for 3 key
    MC_MDA_TONE_4 ,             //DTMF for 4 key
    MC_MDA_TONE_5 ,             //DTMF for 5 key
    MC_MDA_TONE_6 ,             //DTMF for 6 key
    MC_MDA_TONE_7 ,             //DTMF for 7 key
    MC_MDA_TONE_8 ,             //DTMF for 8 key
    MC_MDA_TONE_9 ,             //DTMF for 9 key
    MC_MDA_TONE_A ,             //DTMF for A key
    MC_MDA_TONE_B ,             //DTMF for B key
    MC_MDA_TONE_C ,             //DTMF for C key
    MC_MDA_TONE_D ,             //DTMF for D key
    MC_MDA_TONE_POUND ,         //DTMF for # key
    MC_MDA_TONE_STAR ,          //DTMF for * key
    MC_MDA_NOTE_C4,             // 523.2 Hz    -Piano Notes-
    MC_MDA_NOTE_CS4,            // 554.3 Hz
    MC_MDA_NOTE_D4,             // 587.3 Hz
    MC_MDA_NOTE_DS4,            // 622.2 Hz
    MC_MDA_NOTE_E4,             // 659.2 Hz
    MC_MDA_NOTE_F4,             // 698.5 Hz
    MC_MDA_NOTE_FS4,            // 739.9 Hz
    MC_MDA_NOTE_G4,             // 784.0 Hz
    MC_MDA_NOTE_GS4,            // 830.6 Hz
    MC_MDA_NOTE_A4,             // 440.0 Hz
    MC_MDA_NOTE_AS4,            // 466.1 Hz
    MC_MDA_NOTE_B4,             // 493.8 Hz
    MC_MDA_NOTE_C5,             // 1046.5 Hz
    MC_MDA_NOTE_CS5,            // 1108.7 Hz
    MC_MDA_NOTE_D5,             // 1174.6 Hz
    MC_MDA_NOTE_DS5,            // 1244.3 Hz
    MC_MDA_NOTE_E5,             // 1318.5 Hz
    MC_MDA_NOTE_F5,             // 1397.5 Hz
    MC_MDA_NOTE_FS5,            // 1479.9 Hz
    MC_MDA_NOTE_G5,             // 1568.0 Hz
}
```

```

MC_MDA_NOTE_GS5,          // 1661.2 Hz
MC_MDA_NOTE_A5,           // 880.0 Hz
MC_MDA_NOTE_AS5,          // 932.2 Hz
MC_MDA_NOTE_B5,           // 987.7 Hz
MC_MDA_NOTE_C6,           // 2093.1 Hz
MC_MDA_NOTE_CS6,          // 2217.4 Hz
MC_MDA_NOTE_D6,           // 2349.3 Hz
MC_MDA_NOTE_DS6,          // 2489.1 Hz
MC_MDA_NOTE_E6,           // 2637.5 Hz
MC_MDA_NOTE_F6,           // 2793.7 Hz
MC_MDA_NOTE_FS6,          // 2959.9 Hz
MC_MDA_NOTE_G6,           // 3135.9 Hz
MC_MDA_NOTE_GS6,          // 3322.4 Hz
MC_MDA_NOTE_A6,           // 1760.0 Hz
MC_MDA_NOTE_AS6,          // 1864.7 Hz
MC_MDA_NOTE_B6,           // 1975.5 Hz
MC_MDA_NOTE_C7,           // 4186.0 Hz
MC_MDA_NOTE_A7,           // 3520.0 Hz
MC_MDA_NOTE_AS7,          // 3729.3 Hz
MC_MDA_NOTE_B7            // 3951.0 Hz
} MC_MdaToneType;

```

Represents the enumerated sound scale of the tone note value

● Media handler feature structure

```

typedef enum _MC_MdaDevInfo {
    MC_MDAINFO_STREAM_PLAY          = 0x0001,
    MC_MDAINFO_CALL_BY_REFERENCE    = 0x0002,
    MC_MDAINFO_PAUSE_RESUME         = 0x0004,
    MC_MDAINFO_SEEK                 = 0x0008,
    MC_MDAINFO_STREAM_RECORD        = 0x0010,
    MC_MDAINFO_BALANCE              = 0x0020,
    MC_MDAINFO_MIXING               = 0x0040,
    MC_MDAINFO_MIXING_SYNC          = 0x0080,
    MC_MDAINFO_RECORD               = 0x0100
} MC_MdaDevInfo;

```

Media feature	Description
---------------	-------------

MC_MDAINFO_STREAM_PLAY	The media handler supports playing using the streaming method. In this case, the MC_MDAINFO_CALL_BY_REFERENCE bit should not be set.
MC_MDAINFO_CALL_BY_REFERENCE	The media handler uses media data without copying into the internal buffer. In case this bit is not set, the transmitted data is copied into the internal buffer.
MC_MDAINFO_PAUSE_RESUME	The media handler supports the pause/resume function.
MC_MDAINFO_SEEK	The media handler supports the seek function.
MC_MDAINFO_STREAM_RECORD	The media handler supports recording using the streaming method. In this case, the recorded data can be copied from the internal buffer of the media handler into the platform buffer during recording, and the media handler should be able to record data into the emptied buffer consecutively.
MC_MDAINFO_BALANCE	Set when the media handler supports the left-right sound balance adjustment function; with 50 as the criterion, only the left sound is activated in case of 0 (in case of 100, only the right sound is activated)
MC_MDAINFO_MIXING	The media handler can play multiple media data at the same time. In case simultaneous play is not supported, the corresponding error (M_E_INPROGRESS) should be returned when trying to create multiple media handler instance on the same type of media handler.
MC_MDAINFO_MIXING_SYNC	Refers to the multi channel synchronous playing function; it can be set when playing is available in a state wherein multiple files are synchronized on each channel of the media handler
MC_MDAINFO_RECORD	Refers to supporting recording that is not in the streaming method; in this case, the recorded data cannot be copied from the internal buffer of the media handler into the corresponding buffer during recording, and the data can be copied only after completing the recording procedure

● **Media handler control command**

```
typedef enum _MC_MdaDevControl {
    MC_MDADEVCTRL_GET_INSTANCE_COUNT = 1001,
    MC_MDADEVCTRL_DEVICE_GET_STATUS,
    MC_MDADEVCTRL_DEVICE_DETECT,
    MC_MDADEVCTRL_DEVICE_MODEL,
    MC_MDADEVCTRL_GET_MODE_LIST
} MC_MdaDevControl;
```

This is the Media handler control command enumeration structure used in the MC_mdaClipDevControl() function.

Media handler control commands are commands to be applied by each media handler. The types of media handler control commands that should be supported by each media handler are described at the reference part of MC_mdaClipDevControl().

- **Mode control command**

```
typedef enum _MC_MdaModeControl {
    MC_MDAMODECTRL_GET = 0,
    MC_MDAMODECTRL_SET
} MC_MdaModeControl;
```

Control command used in the MC_mdaClipModeControl() function

- **Property ID used in mode control commands**

Property ID used in the MC_mdaClipModeControl() function

```
typedef enum _MC_MdaModePID{
    MC_MDAMODEPID_N_SAMPLE_PER_SEC,
    MC_MDAMODEPID_N_CHANNELS,
    MC_MDAMODEPID_N_BIT_PER_SAMPLE,
    MC_MDAMODEPID_BALANCE,
    MC_MDAMODEPID_POSITION_X,
    MC_MDAMODEPID_POSITION_Y,
    MC_MDAMODEPID_WIDTH,
    MC_MDAMODEPID_HEIGHT,
    MC_MDAMODEPID_AXIS,
    MC_MDAMODEPID_BRIGHT,
    MC_MDAMODEPID_MAGPOWER,
    MC_MDAMODEPID_RESOLUTION_X,
    MC_MDAMODEPID_RESOLUTION_Y,
    MC_MDAMODEPID_YUV_RESOLUTION_X,
    MC_MDAMODEPID_YUV_RESOLUTION_Y,
    MC_MDAMODEPID_FRAMERATE
} MC_MdaModePID;
```

Property name	Description
MC_MDAMODEPID_N_SAMPLE_PER_SEC	Audio sample speed (default value: 8000KHz)
MC_MDAMODEPID_N_BIT_PER_SAMPLE	Audio sample size (default value: 8 bit)

MC_MDAMODEPID_N_CHANNELS	Number of channels (mono/stereo); the default value is mono (in case of Mono, the value is 1, and Stereo, the value is 2)
MC_MDAMODEPID_BALANCE	With the sound balance 50 as the criterion, the left sound becomes louder in case of a value that is lower than 50; in case of a value that is larger than 50, however, the right sound becomes louder (balance area: 0-100; default value: 50)
MC_MDAMODEPID_POSITION_X	X coordinate of the screen (pixel unit) Default value: 0
MC_MDAMODEPID_POSITION_Y	Y coordinate of the screen (pixel unit) Default value: 0
MC_MDAMODEPID_WIDTH	Screen width (pixel unit); the default value is the width of the whole screen
MC_MDAMODEPID_HEIGHT	Screen height (pixel unit); the default value is the height of the entire screen
MC_MDAMODEPID_AXIS	Rotation/inversion value of the screen (refer to the structure MH_MdaCameraSetAxis regarding the value); the default value is the normal screen
MC_MDAMODEPID_BRIGHTNESS	Brightness of the screen (%; default value: 50)
MC_MDAMODEPID_MAGPOWER	Screen magnification (%; 100 is normal; 200 is x2; 400 is x4; 150 is x1.5); the default value is 100
MC_MDAMODEPID_RESOLUTION_X	Horizontal value of resolution (pixel unit) Resolution default value: 320*240
MC_MDAMODEPID_RESOLUTION_Y	Vertical value of resolution (pixel unit) Resolution default value: 320*240
MC_MDAMODEPID_YUV_RESOLUTION_X	Horizontal value of YUV resolution. (pixel unit) Resolution default value: 320*240
MC_MDAMODEPID_YUV_RESOLUTION_Y	Vertical value of YUV resolution (pixel unit) Resolution default value: 320*240
MC_MDAMODEPID_FRAME_RATE	Number of frames per second (default value: 10)

- **(MC_MdaClip)**

Prototype

typedef void MC_MdaClip

Description

Media clip identifier

- **MC_MDA_STATUS_END_OF_MEDIA**

Prototype

#define MC_MDA_STATUS_END_OF_MEDIA 9

Description

Status wherein the playing for all media data is over (constant value: 9)

- **MC_MDA_STATUS_STOPPED_AT_TIME**

Prototype

```
#define MC_MDA_STATUS_STOPPED_AT_TIME 10
```

Description

Status wherein the media stopped at the stop point set by the MC_MDACTRL_SET_STOP_TIME control command

- **(MEDIACB)**

Prototype

typedef void(*MEDIACB)(MC_MdaClip* clip, M_Int32 status)

Description

Callback function that is called when the state of a media player is changed (for the constant value, refer to the state of media handling)

Parameters

clip Clip
status Status of the media handler

Side Effects

None

Reference Item

None

● MC_mdaClipCreate

Prototype

MC_MdaClip* ***MC_mdaClipCreate(M_Char* mType, M_Int32 bufSize, MEDIACB cb)***

Description

This function creates a specific type of clip. The types supported are those obtained by MEDIADEVICES of MC_knlGetSystemProperty(). In case the type is one that is supported by MIME, the type follows the MIME type, i.e., "audio/xxx" or "video/xxx."

The buffer size of the clip should be as much as the total size of the data.

When the callback function is not registered, the status change of a media handler is not transmitted.

Parameters

[in]	mType	Media type
[in]	bufSize	Buffer size (buffer size to be created within a clip)
[in]	cb	Callback function that will indicate the status change while a clip is handled by the media handler

Return Value

Pass	MC_MdaClip object pointer
Fail	0

Side Effects

None

Reference Item

None

● MC_mdaClipFree**Prototype*****M_Int32 MC_mdaClipFree(MC_MdaClip* clip)*****Description**

Frees all resources allocated to a clip

Parameters

[in] clip Clip

Return Value

Pass

0

Fail

M_E_INUSE

Issued in case of an attempt to free a clip while it is being played or voice recorded

M_E_INVALID

Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipAllocPlayer

Prototype

M_Int32 MC_mdaClipAllocPlayer(MC_MdaClip clip, M_Char* param)*

Description

This function allocates a player that is used to handle the media data within a clip. It should be called before actual handling is effected, such as playing using the media data of a clip. When MC_mdaPlay (for play) is called without calling this function, or after this function failed, an error value is returned.

When media data is directly accessed by a media handler instead of being sent by the application, it is necessary to transmit parameters to a param variable. In this case, there is no need to copy the media data using the MC_mdaClipPutData function. For example, in case of a media handler supporting the streaming play method receiving an argument with url type, the url address and various parameters may be required. Such parameters are transmitted through param (for more information, refer to the Reference Item).

Parameters

[in]	clip	Clip
[in]	param	Required parameter when opening a media handler; when no parameter is required, a null value is inputted The first keyword part of the string passed may vary depending on the parameter required by the device (refer to the table below)

Return Value

Pass		
	0	
Fail		
	M_E_ERROR	Issued when the player allocation fails
	M_E_INVALID	Issued when the clip is null
	M_E_NOTSUP	Issued when the media handler is not supported
	M_E_INPROGRESS	Issued when the maximum instance is exceeded

Side Effects

None

Reference Item

Description on param:

Media Handler	Description
Media handler that can access the URL type streaming media directly	Adds the keyword “-streamURL” before a URL string prior to transmission Ex. 1) -streamURL http://www.media.com/m.mp3 Ex. 2) -streamURL mms://www.media.com/m.mp3 Ex. 3) -streamURL rtsp://www.media.com/m.mp3
Media handler that can access the file directly	Adds the keyword “-file” before a filename prior to transmission Ex. 1) -file media/sample.mp3 Ex. 2) -file sample.mov

● **MC_mdaClipFreePlayer**

Prototype

M_Int32 MC_mdaClipFreePlayer(MC_MdaClip clip)*

Description

Frees the player that was allocated using the MC_mdaClipAllocPlayer() method; when this function is called while the MC_mdaClipFree() function is not called, the player allocated to this clip should be automatically freed

Parameters

[in] clip Clip

Return Value

Pass	0	
Fail	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipGetType

Prototype

M_Int32 MC_mdaClipGetType(MC_MdaClip* clip, M_Char* buf, M_Int32 bufSize)

Description

Gets the clip type

Parameters

[in]	clip	Clip
[out]	buf	Buffer where a type is stored
[in]	bufSize	Size of a buffer to be copied

Return Value

Pass	0	
Fail	M_E_SHORTBUF	Issued when the buffer to be stored is too small
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipGetInfo

Prototype

M_Int32 MC_mdaClipGetInfo(MC_MdaClip clip, M_Int32 *rtnInfo)*

Description

Gets the media properties that can be supported from a media handler supported by the terminal depending on the media type of a clip.

Parameters

[in]	clip	Clip
[out]	rtnInfo	OR operation value of the Bit Mask for the structure of media handler properties (MC_MdaDevInfo)

Return Value

Pass	0	
Fail	M_E_ERROR	Issued when the media property cannot be obtained
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

Media handler properties structure (MC_MdaDevInfo)

● MC_mdaClipDevControl

Prototype

M_Int32 MC_mdaClipDevControl(MC_MdaClip* clip, MC_MdaDevControl cmd, void* buf1, void* buf2)

Description

This function is used when implementing device function commands supported by the manufacturer in addition to the common functions (play, stop, pause) of the media. For example, if the manufacturer supports a command that turns on a camera, such command can be implemented using this function. When a control command supported by the manufacturer should be implemented by device, this function is used. Commands belonging to this category include commands that turn on and turn off power. On the other hand, when a control command supported by the manufacturer should be implemented by media content, the MC_mdaClipControl() function is used. Commands belonging to this category include a command that gets the current play time of a media content that is currently being played.

Parameters

[in]	clip	Clip
[in]	cmd	Control command
[in]	buf1	buf1 that can be used in control command
[in]	buf2	buf2 that can be used in control command

Return Value

Pass	0
Fail	M_E_ERROR
	M_E_INVALID
	Issued when the clip is null

Side Effects

None

Reference Item

1. Media device control commands that can be supported by clip type

MIME Type	Supported Media Device Control Command
Qualcomm_CMX	MC_MDADEVCTRL_GET_INSTANCE_COUNT, // maximum number of instances that can be supported MC_MDADEVCTRL_GET_MODE_LIST // getting the supported mode names
Yamaha_MA1	
Yamaha_MA2	
Yamaha_MA3	
Yamaha_MA5	
Yamaha_SMAF	
Yamaha_SMAF-Phrase	
Yamaha_SMAF-Audio	

audio/MIDI	
audio/WAVE	
audio/MP3	
audio/TONE	
audio/FREQTONE	
IS96	
IS96A	
IS733	
IS127	
G.723.1	
Audio/AAC	
Audio/AAC+	
Video/MPEG4	MC_MDADEVCTRL_GET_INSTANCE_COUNT, // maximum number of instances that can be supported MC_MDADEVCTRL_DEVICE_GET_STATUS, // gets the power status of a camera MC_MDADEVCTRL_DEVICE_DETECT, // detects the mounting status of a camera MC_MDADEVCTRL_DEVICE_MODEL, // gets the model name of a camera
Video/H.263	
Video/H.264	
Video/MJPEG	
Image/JPEG	

2. Description on media handler control commands and parameters

cmd	MC_MDADEVCTRL_GET_INSTANCE_COUNT
buf1	None
buf2	[out] *(M_Int32*) buf2: Number of instances supported
Description	Gets the number of instances supported by this device
Remarks	At least one instance should be supported.
cmd	MC_MDADEVCTRL_DEVICE_GET_STATUS
buf1	[out] *(M_Boolean*) buf1 – Media handler power ON/ OFF
buf2	None
Description	This determines the power status of the media handler. When power is ON, transmit TRUE to buf1; when power is OFF, transmit FALSE.
Remarks	
cmd	MC_MDADEVCTRL_DEVICE_DETECT
buf1	None
buf2	[out] Pass: *(M_Int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR
Description	Detects internal and external cameras
Remarks	
cmd	MC_MDADEVCTRL_DEVICE_MODEL
buf1	[in] *(M_Int32*) buf1: Length of a camera model name
buf2	[out] Pass: (M_Char*) buf2 = Camera model name Fail: *(M_Int32) buf2 = M_E_LONGNAME / Issued when the length of the model name is too long *(M_Int32) buf2 = M_E_ERROR / Issued when the model name cannot be obtained
Description	Gets the model name of a camera
Remarks	
Remarks	At least one model should be supported.
cmd	MC_MDADEVCTRL_GET_MODE_LIST
buf1	[in] (M_Int32*) buf1[0] – Buffer size that receives the mode names
buf2	[out] (M_Char*) buf2 – Mode names supported by the manufacturer
Description	This retrieves the list of mode names supported by the manufacturer. Mode pertains to the structure composed of the general properties of the media handler. To determine or modify the property value, the MC_mdaClipModeControl() function can be used. The general properties of each media handler was already defined. Additional specific properties can be used for the mobile communication service provider or the manufacturer. In case of various modes supported, insert “,” between the mode names to distinguish one from the other. At least 1 mode should be supported, and the mode name should be “DEFAULT_MODE.” Examples of mode names that can be transmitted to buf2 include “DEFAULT_MODE, SKT_MODE, and LG_MODE.”

Remarks	
---------	--

● MC_mdaClipControl

Prototype

M_Int32 MC_mdaClipControl(MC_MdaClip* clip, M_Int32 cmd, void* buf1, void* buf2)

Description

This function is used when implementing special function commands supported by the manufacturer in addition to the common functions (play, stop, pause) of the media. For example, if the manufacturer supports a control command that can get the current play time, such command can be implemented using this function.

Parameters

[in]	clip	Clip
[in]	cmd	Control command
[in]	buf1	buf1 that can be used in the control command
[in]	buf2	buf2 that can be used in the control command

Return Value

Pass	0	
Fail	M_E_ERROR	
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

1. Media control commands that can be supported by clip type

MIME Type	Supported Media Control Command
Qualcomm_CMX	MC_MDACTRL_GET_MEDIA_TIME, // current play time of the media MC_MDACTRL_SET_SYNC, // sets synchronization between media MC_MDACTRL_GET_SYNC, // gets synchronized media. MC_MDACTRL_GET_STOP_TIME // determines the stop point for playing MC_MDACTRL_SET_STOP_TIME // sets the stop point for playing MC_MDACTRL_SET_MODE // sets the mode after getting the name
Yamaha_MA1	
Yamaha_MA2	
Yamaha_MA3	
Yamaha_MA5	
Yamaha_SMAF	
Yamaha_SMAF-Phrase	
Yamaha_SMAF-Audio	
"audio/ONEPOLY"	
"audio/GVMONEPOLY"	
audio/MIDI	
audio/WAVE	
audio/MP3	
audio/TONE	
audio/FREQTONE	
IS96	
IS96A	
IS733	

IS127	
G.723.1	
audio/AAC	
audio/AAC+	
video/MPEG4	MC_MDACTRL_GET_MEDIA_TIME, // current play time of the media
VOD_URL	
video/H.263	
video/H.264	
video/mjpeg	MC_MDACTRL_CAPTURE_IMAGE, // captures a stopped image MC_MDACTRL_GET_CAPTURE_IMAGE // retrieves captured still image data MC_MDACTRL_GET_STOP_TIME // determines the stop point of playing MC_MDACTRL_SET_STOP_TIME // sets the stop point of playing MC_MDACTRL_SET_MODE // receives mode name and sets MC_MDACTRL_PREVIEW_START, // starts camera preview MC_MDACTRL_PREVIEW_STOP, // stops camera preview
image/jpeg	

2. Description on media control commands and parameters

cmd	MC_MDACTRL_GET_MEDIA_TIME
buf1	None
buf2	[out] Pass: *(M_int32*) buf2 = Current play time (millisecond) Fail: *(M_Int32*) buf2 = M_E_ERROR Issued in case of other errors *(M_Int32*) buf2 = M_E_NOTSUP Issued when this command is not supported
Description	Gets the current play time vis-à-vis the total play time (millisecond)
Remarks	
cmd	MC_MDACTRL_SET_SYNC
buf1	[in] *(M_Int32*) buf1[0] = Array size of an instance ID of the media device *(M_Int32*) buf1[1] = First slave media instance ID to be synchronized *(M_Int32*) buf1[2] = Second slave media instance ID to be synchronized To be repeated as much as the array size
buf2	[out] Pass: *(M_int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR Issued in case of other errors occur *(M_Int32*) buf2 = M_E_NOTSUP Issued when this command is not supported
Description	Sets channel synchronization between media played in multi-channels; to release synchronization, 0 should be passed to *(M_Int32*) buf1[0]
Remarks	
cmd	MC_MDACTRL_GET_SYNC
buf1	[in] *(M_Int32*) buf1 = Maximum size of a multi-channel array
buf2	[out]

	Pass: *((M_Int32*)buf2+0) = First synchronized slave media instance ID *((M_Int32*)buf2+1) = First synchronized slave media instance ID ... To be repeated as much as the array size Fail: *((M_Int32*)buf2) = M_E_NOTSUP: Issued when synchronization is not supported *((M_Int32*)buf2) = M_E_ERROR: Issued in case of other errors
Description	Gets channel synchronization information between media played in multi-channels
Remarks	
cmd	MC_MDACTRL_SET_STOP_TIME (millisecond)
buf1	[in] *((M_Int32*) buf1) = Time when to stop playing (millisecond)
buf2	[out] Pass: *((M_Int32*) buf2) = 0 Fail: *((M_Int32*) buf2) = M_E_NOTSUP: Issued when this command is not supported *((M_Int32*) buf2) = M_E_ERROR: Issued in case of other errors
Description	Sets the time when playing should be stopped vis-à-vis the total play time of the media
Remarks	
cmd	MC_MDACTRL_GET_STOP_TIME
buf1	[out] (M_Int32*)buf1 – Time period to set stopped
buf2	None
Description	Determines the temporary stop point currently set in milliseconds; when the temporary stop point is not set, a value of 1 is received for parameter buf1
Remarks	
cmd	MC_MDACTRL_SET_MODE
buf1	[in] *((M_Char*) buf1) : Mode name
buf2	[out] Pass : *((M_Int32*) buf2) = 0 Fail : *((M_Int32*) buf2) = M_E_ERROR Other errors *((M_Int32*) buf2) = M_E_NOTSUP Unsupported mode names *((M_Int32*) buf2) = M_E_INVALID Incorrect mode names
Description	Set mode by named buf1
Remarks	
Cmd	MC_MDACTRL_CAPTURE_IMAGE
buf1	[in] (M_Char*) buf1 = Buffer where a captured screen shot is to be stored
buf2	[in] *((M_Int32*) buf2) = Size of a buffer where a captured screen shot is to be stored
Description	Captures a screen shot of a video being played
Remarks	Return value Pass: Size of a screen shot captured Fail: M_E_NOTSUP: Issued when this command is not supported M_E_ERROR: Issued in case of other errors
cmd	MC_MDACTRL_GET_CAPTURE_IMAGE
buf1	[in] (M_Int32*)buf1[0] – Size of the image data
buf2	[out] (M_Uint8*)buf2 – Buffer where the captured image is saved
Description	Get image data by using MC_MDACTRL_CAPTURE_IMAGE

	control command.
Remarks	return value Pass : Size of the captured screenshot Fail : _E_NOTSUP : Not supported M_E_ERROR : Other errors
Cmd	MC_MDACTRL_PREVIEW_START
buf1	None
buf2	[out] Pass: *(M_Int32*) buf2 = 0 Fail: *(M_Int32*) buf2 = M_E_ERROR Issued in case of other errors
Description	Starts preview play based on the currently set display mode and display size; when the preview play is in progress, this command does not play any role
Remarks	
cmd	MC_MDACTRL_PREVIEW_STOP
buf1	None
buf2	None
Description	Stops preview play while it is in progress; when the preview play is not in progress, this command does not play any role
Remarks	This function should succeed without any condition.

● MC_mdaClipGetModeList

Prototype

M_Int32 MC_mdaClipGetModeList(MC_MdaClip clip, M_Char* modeList, M_Int32 size)*

Description

This function gets a list of mode names supported by the terminal. A mode refers to a structure that is a collection of data values for the attributes of a clip. The manufacturer or a mobile communication service provider is required to provide support for one or more modes. The name of a required mode is DEFAULT_MODE, whose attributes can be obtained and modified using the control command of the MC_mdaClipModeControl() function. When supporting other modes with mode ID name other than DEFAULT_MODE, the manufacturer or a mobile communication service provider should provide information on the internal attribute values of the mode to CP.

When several modes are supported, the comma (,) will serve as the separator between mode names, e.g., DEFAULT_MODE, SKT_MODE, LG_MODE.

Parameters

[in]	clip	Clip
[out]	modeList	Name list of modes supported by the manufacturer; mode names are separated by a comma (,)
[in]	size	Size of a buffer from where a name list of modes supported by the manufacturer is to be received

Return Value

Pass	0	
Fail	M_E_ERROR	
	M_E_INVALID	Issued when the clip is null
	M_E_SHORTBUF	Issued when the size of the buffer is too small

Side Effects

None

Reference Item

None

● MC_mdaClipModeControl

Prototype

M_Int32 MC_mdaClipModeControl(MC_MdaClip* clip, M_Char* modeName, MC_MdaModeControl cmd, MC_MdaModePID pID, void* buf)

Description

This function applies a control command for attribute information constituting a specific mode of a clip. The following is the type of mode attribute information that can be supported depending on the clip type. When an attempt is made for a command control for information on mode attributes that are not supported by the clip, the corresponding error value or M_E_ERROR is returned. When modifying attributes of a currently set mode, the modification is applied immediately. On the other hand, when modifying the attributes of other modes, the MC_MDACTRL_SET_MODE control command of the MC_mdaClipControl() function should be called using the mode name as a parameter.

Parameters

[in]	clip	Clip
[in]	modeName	Mode name; a list of mode names supported by the terminal can be obtained through the MC_mdaClipGetModeList function (among these mode names, modeName refers to the mode name from which attribute data values are to be obtained or modified)
[in]	cmd	Control command. MC_MDAMODECTL_GET/MC_MDAMODECTL_SET MC_MDAMODEPID_BALANCE
[in]	pID	Attribute ID used to implement the control command
[in/out]	buf	buf that can be used in the control command
	[in]	when the control command is MC_MDAMODECTL_SET,
	[out]	when the control command is MC_MDAMODECTL_GET

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case of other errors
	M_E_NOTSUP	Issued when the mode setting information is not supported by the clip type or the terminal

M_E_INVALID

Issued when the clip is null

Side Effects

None

Reference Item

1. Rules on Mode Name

All modes should have mode names that are distinguishable from other modes. A mode name is not to be determined by the developer at random; rather, it should comply with the following rules:

- ① At least one or more mode should be supported, including a mode whose name is DEFAULT_MODE.
- ② In case of a mode that is provided by a mobile communication service provider or the manufacturer, it will follow the following rules:

(Name of the mobile communication service provider or manufacturer)_MODE_(Index)

Ex.) In case SKT supports three modes, the names of the three modes shall be SKT_MODE_0, SKT_MODE_1, and SKT_MODE_2.

2. Attribute Argument of DEFAULT_MODE

A mode named DEFAULT_MODE should be supported by the terminal without any condition. This mode consists of common attribute arguments held by each media handler.

3. Attribute arguments held by DEFAULT_MODE by media type

Media Handler		Attribute Information
ringtone	"audio/ONEPOLY"	Balance Tempo Pitch
	"audio/GVMONPOLY"	
	audio/MIDI	
	audio/TONE	
	audio/FREQTONE	
	Qualcomm_CMX	
	Yamaha_MA1	Balance
	Yamaha_MA2	
	Yamaha_MA3	
	Yamaha_MA5	
	Yamaha_SMAF	
	Yamaha_SMAF-Phrase	
	Yamaha_SMAF-Audio	
vocoder	IS96	Balance
	IS96A	
	IS733	
	IS127	
	G.723.1	
	AMR-WB	
	AMR-NB	
general sound	audio/WAVE	Sample per second Significant bits per sample Number of channels
	audio/MP3	
	audio/AAC	

	audio/AAC+	Balance
video	video/MPEG4	Location (x position, y position)
	"VOD_URL"	Size (width, height)
	video/H.263	Axis
	video/H.264	Bright
video capture	video/MJPEG	MagPower
	image/JPEG	Resolution (x , y) YUV Resolution (x , y) FrameRate

4. Detailed description on attribute IDs to be held by DEFAULT_MODE

Media Handler	Attribute Name	Description
ringtone		
vocoder		
	Sample per second	Audio sample speed; default value is 8000 KHz
	Significant bits per sample	Audio sample size; default value is 8 bit
	Number of channels	Number of channels (Mono / Stereo); default value is mono
	Balance	This is the sound balance. Taking 50 as the reference, when the value is smaller than 50, the left sound becomes larger; when the value is larger than 50, the right sound becomes larger. The range of balance value is between 0 and 100. The default value is 50.
video video capture	Location (x position, y position)	X coordinate and Y coordinate of the display (pixel); the default value for the X coordinate is 0, whereas the default value for the Y coordinate is 0
	Size (width, height)	Width and height of the display (pixel); the default value is the entire display size
	Axis	Rotation/inversion value of the display (for rotation/inversion value of the display, refer to structure MH_MdaCameraSetAxis); the default value is normal display
	Bright	Brightness of the display (percent); the default value is 50.
	MagPower	Magnification rate (percent; when magnification rate is 100, common rate; when the rate is 200, 2x; when the rate is 400, 4x; when the rate is 150, 1.5x); the default value is 100
	Resolution (x , y)	Width and length values of the resolution (pixel); the default value is 320*240
	YUV Resolution (x , y)	Width and length values of the YUV resolution (pixel); the default value is 320*240
	FrameRate	Number of frames per second; the default value is 10

5. Data type on attribute IDs to be held by DEFAULT_MODE

Attribute name	Data Type
Sample Per Second	M_Int32
Significant bits per sample	M_Int32
Number of Channels	M_Int32

Balance	M_Int32
Location (x position, y position)	M_Int32, M_Int32
Size(width, height)	M_Int32
Axis	M_Int32
Bright	M_Int32
MagPower	M_Int32
Resolution (x , y)	M_Int32
YUV Resolution (x , y)	M_Int32
FrameRate	M_Int32

● MC_mdaClipPutData

Prototype

M_Int32 MC_mdaClipPutData(MC_MdaClip clip, M_Byte* buf, M_Int32 size)*

Description

This function copies media data to a clip when the media data to be inputted is stored in the memory. The media data should have a type that has been registered when the clip was created. Data within a clip is reduced when it is played in the media player and increased through MC_mdaClipPutData(). When the data size to be copied is bigger than what can be accommodated by the internal buffer of the clip, only the portion of the data that can be accommodated is copied.

Parameters

[in]	clip	Clip
[in]	buf	Direct buffer
[in]	size	Size of the buffer to be copied

Return Value

Pass	Size copied
Fail	
M_E_INVALID	Issued when CLIP is NULL
M_E_ERROR	Issued when copying media data failed

Side Effects

None

Reference Item

None

● MC_mdaClipGetData

Prototype

M_Int32 MC_mdaClipGetData(MC_MdaClip clip, M_Byte* buf, M_Int32 size)*

Description

This function copies media data from a clip to the buffer. Data within a clip is increased when it is recorded in the media player and reduced in MC_mdaClipGetData(). When the data within a clip is bigger than the buffer transmitted, only the portion that fits the buffer size is copied. This function is used when the clip type is what has been obtained in MEDIADEVICES.

Parameters

[in]	clip	Clip
[in]	buf	Direct buffer
[in]	size	Size of the buffer to be copied

Return Value

Pass	Size copied	
Fail	M_E_INVALID	Issued when CLIP is NULL
	M_E_ERROR	Issued when copying media data failed

Side Effects

None

Reference Item

None

- **MC_mdaClipAvailableDataSize**

Prototype

M_Int32 MC_mdaClipAvailableDataSize(MC_MdaClip* clip)

Description

Return the data size that is available to a clip (not the buffer size within a clip)

Parameters

[in] clip Clip

Return Value

Pass

Available data size

Fail

M_E_INVALID

Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipClearData

Prototype

M_Int32 MC_mdaClipClearData (MC_MdaClip clip)*

Description

Clears all available data within a clip

Parameters

[in] clip Clip

Return Value

Pass

0

Fail

M_E_ERROR

Issued when this function is called during play or pause

M_E_INVALID

Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipSetPosition

Prototype

M_Int32 MC_mdaClipSetPosition(MC_MdaClip clip, M_Int32 ms)*

Description

Sets the position where play is to begin; when this function is called to a clip that was created with a type that does not support the play position setting function, an error or M_E_NOTSUP is returned

Parameters

[in]	clip	Clip
[in]	ms	Time when clip play will start (millisecond)

Return Value

Pass	0	
Fail	M_E_NOTSUP	Issued when the clip player does not support the play position setting function
	M_E_ERROR	Issued when this function is called in case the media handler is not in temporary stop status
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipGetVolume

Prototype

M_Int32 MC_mdaClipGetVolume(MC_MdaClip clip)*

Description

This function reads the volume of a clip player. When the independent volume setting of a clip player is supported, this function reads the volume of a clip player. On the other hand, when independent volume setting is not supported, this function can indicate an identical volume source even if the clip creation type is different.

The minimum value of the volume is 0, and the maximum value, 100.

Parameters

[in] clip Clip

Return Value

Pass	Volume value
Fail	
M_E_NOTSUP	Issued when there is no volume value for the media handler
M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaClipSetVolume

Prototype

M_Int32 MC_mdaClipSetVolume(MC_MdaClip clip, M_Int32 level)*

Description

This function sets the volume of a clip player. When the independent volume setting of a clip player is supported, this function sets the volume of a clip player. On the other hand, when independent volume setting is not supported, this function can indicate an identical volume source even if the clip creation type is different.

The minimum value of the volume is 0, and the maximum value, 100.

Parameters

[in]	clip	Clip
[in]	level	Volume value (0 - 100)

Return Value

Pass		
	0	
Fail		
	M_E_NOTSUP	Issued when the media handler does not support volume value setting
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaPlay

Prototype

M_Int32 MC_mdaPlay(MC_MdaClip clip, M_Boolean repeat)*

Description

This function plays the data of a clip.

When this function is called, and media handling starts, MC_MDA_STATUS_STARTED is transmitted to the callback function registered when the clip was created. In case this function is already called, and media handling is in progress, this function does not play any role. If there is no more clip data, MC_MDA_STATUS_END_OF_DATA is transmitted to the callback function.

When playing using the streaming method, the MC_MDA_STATUS_END_OF_DATA status is transmitted to the callback function before the clip data is totally used (when getting the water mark). At this time, in case there is remaining data to be played, the clip data should be filled using the MC_mdaClipPutData() function.

Parameters

[in]	clip	Clip
[in]	repeat	Once when the parameter is 0; repeat play when the parameter is 1

Return Value

Pass	0	
Fail		
	M_E_INUSE	Issued when the clip player is already playing another clip
	M_E_ERROR	Issued when the same clip is already being played
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaPause

Prototype

M_Int32 MC_mdaPause(MC_MdaClip clip)*

Description

This function temporarily stops media handling (play/recording).

When this function is called, and media handling is temporarily stopped, MC_MDA_STATUS_PAUSE is transmitted to the callback function registered when the clip was created. When this function is called again to a paused or stopped media handler, this function does not play any role.

Parameters

[in] clip Clip

Return Value

Pass	0	
Fail	M_E_NOTSUP	Issued when the clip player does not support 'pause'
	M_E_ERROR	Issued when the media handler is already paused or stopped
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaResume

Prototype

M_Int32 MC_mdaResume(MC_MdaClip clip)*

Description

This function resumes media handling (play/recording) that was temporarily stopped.

When this function is called, and media handling is resumed, MC_MDA_STATUS_RESUME is transmitted to the callback function registered when the clip was created. When this function is called again to the media handler that is handling media, this function does not play any role.

Parameters

[in] clip Clip

Return Value

Pass	0	
Fail	M_E_NOTSUP	Issued when the clip player does not support 'resume'
	M_E_ERROR	Issued when the media handler is already handling media
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaStop

Prototype

M_Int32 MC_mdaStop(MC_MdaClip clip)*

Description

This function stops media handling (play/recording).When this function is called, and media handling is stopped, MC_MDA_STATUS_STOP is transmitted to the callback function registered when the clip was created. When this function is called again to a stopped media handler, this function does not play any role.

Parameters

[in] clip Clip

Return Value

Pass	Returns 0 during playing or stop status	
	Returns the recorded amount during recording	
Fail	M_E_ERROR	Issued when the media handler is already stopped
	M_E_INVALID	Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaRecord

Prototype

M_Int32 MC_mdaRecord(MC_MdaClip clip)*

Description

This function starts recording.

When an attempt is made to record using a clip that was created with a type that does not support recording, this function does not play any role. When this function is called, and media handling starts, MC_MDA_STATUS_RECORD is transmitted to the callback function registered when the clip was created. In case this function is called, and recording is in progress, this function does not play any role. When the internal buffer of the clip is completely filled during recording, MC_MDA_STATUS_END OF MEDIA is transmitted to the callback function.

When recording using the streaming method, the clip internal buffer should be emptied using MC_mdaClipGetData() before the clip internal buffer is completely filled (when getting the water mark).

Parameters

[in] clip Clip

Return Value

Pass

0

Fail

M_E_INUSE

Issued when there is another clip that is already recording

M_E_ERROR

Issued when the same clip is being recorded

M_E_INVALID

Issued when the clip is null

Side Effects

None

Reference Item

None

● MC_mdaGetVolume

Prototype

M_Int32 MC_mdaGetVolume()

Description

This function returns the volume value. When an independent volume is set to each media handler, this value may not be accurate. In such cases, the exact volume of each handler should be read using MC_mdaClipGetVolume (). The volume value to be returned should be converted to a value between 0 and 100 before it is returned. The degree of volume loudness that should be matched by the value between 0 and 100 shall be decided based on the percentage of the volume stage supported by the hardware as shown in the following example. The number of stages of volume loudness supported by the hardware will be returned through MC_knlGetSystemProperty().

Ex.) Hardware with two volume loudness (loud, weak) => 1-50: weak volume; 51-100: loud volume

Hardware with three volume loudness (loud, medium, weak) => 1-33: weak volume; 34-66: medium volume; 67-100: loud volume

Parameters

None

Return Value

Pass

Volume value

Fail

M_E_NOTSUP

Issued when the volume setting is not supported

Side Effects

None

Reference Item

None

- **MC_mdaSetVolume**

Prototype

void MC_mdaSetVolume(M_Int32 value)

Description

This function sets the volume for all media handlers for which volume can be set. When setting the independent volume for each handler, volume API in the clip should be used. The minimum volume value to be set is 0, and the maximum volume value, 100.

Parameters

[in] value Volume value (0 - 100)

Return Value

None

Side Effects

None

Reference Item

None

● MC_mdaVibrator

Prototype

M_Int32 MC_mdaVibrator(M_Int32 level, M_Int32 timeout)

Description

This function controls the vibrator. The vibrator is on for a specified time, and goes off automatically.

Only when the parameter level value is larger than 0 will the timeout value be valid. A level value of 0 means that the vibrator is turned off. Vibration strength is determined by the level value, which can be a value between 0 and 100. 100 represents the strongest vibration supported by the hardware, and 0, the weakest vibration. The degree of vibration strength that should be matched by the value between 0 and 100 shall be decided based on the percentage of the vibration stage supported by the hardware as shown in the following example. The number of stages of vibration strength supported by the hardware can be determined through MC_knlGetSystemProperty("VIBRATORLEVEL").

Ex.) Hardware with one vibration strength => 1-100: Vibration

Hardware with two vibration strengths (strong, weak) => 1-50: weak vibration, 51-100: strong vibration

Hardware with three vibration strengths (strong, medium, weak) => 1-33: weak vibration; 34-66: medium vibration; 67-100: strong vibration

Parameters

[in]	level	A value of 0 means that the vibrator is off; When the value is between 1 and 100, the vibrator is turned on, with the vibration strength set by the operating system.
[in]	timeout	Vibration time (millisecond)

Return Value

Pass	0	
Fail	M_E_INUSE	Issued when the vibrator requested is currently in use
	M_E_ERROR	Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

● MC_mdaSetMuteState

Prototype

M_Int32 MC_mdaSetMuteState(M_Char *strCategory, M_Boolean bmute)

Description

Sets the mute state by volume category of a terminal; the volume categories supported by a terminal can be classified as follows:

Category String	Description	Default Volume Example
GENERAL	Used for general applications	Application volume level of the terminal
VOICE	Voice Play/Record feature	Calling volume of the terminal
RING	This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.	Ringing volume of the terminal
KEY	Key tone feature	Key tone volume of the terminal
MESSAGE	SMS message reception alert feature	SMS message volume
ALARM	Alarm alert feature	Alarm volume
ALERT	Various alert features, e.g., no service, low battery, etc.	Alert volume
MMEDIA	This is the volume for the multimedia device. It indicates the master volume of all multimedia devices supported by the platform. The master volume affects all multimedia devices. To set the volume by media device, use the MH_mdaGetVolume/MH_mdaSetVolume function.	Multimedia volume
GAME	Feature used when playing a game	Game volume
OEM	Used when setting a volume level other than those defined above	Other volume

Parameters

[in] strCategory Category string of the default volume
 [in] bmute Mute state setting
 TRUE Mute
 FALSE Not mute

Return Value

Pass

0

Fail

M_E_ERROR

Issued in case of failure

M_E_INVALID

Issued when the category ID does not exist

Side Effects

None

Reference Item

None

● MC_mdaGetMuteState

Prototype

M_Int32 MC_mdaGetMuteState(M_Char * strCategory)

Description

Gets the mute state by volume category of a terminal; the volume categories supported by a terminal can be classified as follows:

Category String	Description	Default Volume Example
GENERAL	Used for general applications	Application volume level of the terminal
VOICE	Voice Play/Record feature	Calling volume of the terminal
RING	This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.	Ringing volume of the terminal
KEY	Key tone feature	Key tone volume of the terminal
MESSAGE	SMS message reception alert feature	SMS message volume
ALARM	Alarm alert feature	Alarm volume
ALERT	Various alert features, e.g., no service, low battery, etc.	Alert volume
MMEDIA	This is the volume for the multimedia device. It indicates the master volume of all multimedia devices supported by the platform. The master volume affects all multimedia devices. To set the volume by media device, use the MH_mdaGetVolume/MH_mdaSetVolume function.	Multimedia volume
GAME	Feature used when playing a game	Game volume
OEM	Used when setting a volume level other than those defined above	Other volume

Parameters

[in] strCategory Category string of default volume

Return Value

Pass

M_E_SUCCESS Mute success

Fail

M_E_ERROR Mute failure

M_E_NOTSUP Issued when the default volume value

setting is not supported

M_E_INVALID Issued when the category ID does not exist

Side Effects

None

Reference Item

None

● MC_mdaSetDefaultVolume

Prototype

M_Int32 MC_mdaSetDefaultVolume(M_Char *strCategory, M_Int32 vol)

Description

This function sets the default volume in a terminal. Default volume setting is possible only in an application with the security level of CP (Contents Provider).

The volume categories supported by a terminal can be classified as follows: The category strings of the default volume supported by terminal can be obtained by calling the MC_knlGetSystemProperty() API의 "DEFAULTVOLUME" command

Category String	Description	Default Volume Example
GENERAL	Used for general applications	Application volume level of the terminal
VOICE	Voice Play/Record feature	Calling volume of the terminal
RING	This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.	Ringing volume of the terminal
KEY	Key tone feature	Key tone volume of the terminal
MESSAGE	SMS message reception alert feature	SMS message volume
ALARM	Alarm alert feature	Alarm volume
ALERT	Various alert features, e.g., no service, low battery, etc.	Alert volume
MMEDIA	This is the volume for the multimedia device. It indicates the master volume of all multimedia devices supported by the platform. The master volume affects all multimedia devices. To set the volume by media device, use the MH_mdaGetVolume/MH_mdaSetVolume function.	Multimedia volume
GAME	Feature used when playing a game	Game volume
OEM	Used when setting a volume level other than those defined above	Other volume

Parameters

[in] strCategory Category string of the default volume
 [in] vol Volume value (0 - 100)

Return Value

Pass

0

Fail

M_E_NOTSUP

Issued when the default volume value
setting is not supported

M_E_ERROR

Issued in case of other errors

M_E_INVALID

Issued in case of invalid parameter

Side Effects

None

Reference Item

None

● MC_mdaGetDefaultVolume

Prototype

M_Int32 MC_mdaGetDefaultVolume(M_Char *strCategory)

Description

Gets the default volume set by a terminal;

the volume categories supported by a terminal can be classified as follows:

These classifications can vary by terminal manufacturer or terminal model. The strings that can be used as the categories of the default volume can be obtained by calling the MC_knlGetSystemProperty API using the "DEFAULTVOLUME" command.

Category String	Description	Default Volume Example
GENERAL	Used for general applications	Application volume level of the terminal
VOICE	Voice Play/Record feature	Calling volume of the terminal
RING	This is the ringing feature, e.g., in case the current ringing mode is set to vibration, vibration occurs during playing instead of sounds. In case of a separate melody speaker, sounds occur through it. It is the same as the feature of a terminal receiving a call.	Ringing volume of the terminal
KEY	Key tone feature	Key tone volume of the terminal
MESSAGE	SMS message reception alert feature	SMS message volume
ALARM	Alarm alert feature	Alarm volume
ALERT	Various alert features, e.g., no service, low battery, etc.	Alert volume
MMEDIA	This is the volume for the multimedia device. It indicates the master volume of all multimedia devices supported by the platform. The master volume affects all multimedia devices. To set the volume by media device, use the MH_mdaGetVolume/MH_mdaSetVolume function.	Multimedia volume
GAME	Feature used when playing a game	Game volume
OEM	Used when setting a volume level other than those defined above	Other volume

Parameters

[in] strCategory Category string of the default volume

Return Value

Pass

Volume value (0 - 100)

Fail

M_E_NOTSUP
is unsupported

Issued in case the default volume function

M_E_ERROR
reason

Issued in case of failure due to other

M_E_INVALID

Issued in case of invalid parameter

Side Effects

None

Reference Item

None

● MC_mdaSetWaterMark

Prototype

M_Int32 MC_mdaSetWaterMark(MC_MdaClip clip, M_Int32 waterMark)*

Description

In case the media handler supports playing and recording through the streaming method, the water mark where MC_MDA_STATUS_END_OF_DATA and MC_MDA_STATUS_FULL_OF_DATA occur should be set. In case the water mark is set to 90% during playing, when more than 90% of the clip buffer is played, the MC_MDA_STATUS_END_OF_DATA event occurs. During recording, when more than 90% of the clip buffer is filled, MC_MDA_STATUS_FULL_OF_DATA occurs. The default value is 100%.

Parameters

[in]	clip	Clip
[in]	watermark	Water mark (0 – 100)

Return Value

Pass	Water mark set
Fail	
	M_E_NOTSUP Issued in case of unsupported function
	M_E_ERROR Issued in case of failure due to unknown reason
	M_E_INVALID Issued when CLIP is NULL

Side Effects

None

Reference Item

None

● **MC_mdaClipPutToneData**

Prototype

M_Int32 MC_mdaClipPutToneData (MC_MdaClip clip,
MC_MdaToneType tone[], M_Int duration[], M_Int32 number)*

Description

This copies the media data into the clip when the clip type is "audio/TONE." The media data should be the type set when creating clip. When playing on the media handler, the data in the clip decrease and increase with MC_mdaClipPutToneData(). When the size of the data to be copied is bigger than the clip internal buffer capacity, only the available amount is copied.

Parameters

[in]	clip	Clip
[in]	tone	Arrangement of tones to be played
[in]	duration	Time arrangement of tones to be played
[in]	number	Number of data to be copied

Return Value

Pass	Number of data copied
Fail	None

Side Effects

None

Reference Item

None

● MC_mdaClipPutFreqToneData

Prototype

M_Int32 MC_mdaClipPutFreqToneData (MC_MdaClip* clip, M_Int32 hiFreq[], M_Int32 lowFreq[], M_Int32 duration, M_Int32 number)

Description

This copies the media data into the clip when the clip type is "audio/FREQTONE." The media data should be the type set when creating clip. When playing on the media handler, the data in the clip decrease and increase with MC_mdaClipPutFreqToneData(). When the size of the data to be copied is bigger than the clip internal buffer capacity, only the available amount is copied.

Parameters

[in]	clip	Clip
	[in] hiFreq	HZ arrangement of high frequency to be played
	[in] lowFreq	HZ arrangement of low frequency to be played
	[in] duration	Time arrangement of tones to be played
[in]	number	Number of data to be copied

Return Value

Pass	Number of data copied	
Fail		
	M_E_ERROR	Issued in case of error
	M_E_INVALID	Issued when CLIP is NULL

Side Effects

None

Reference Item

None

● MC_mdaEnableOEMDisplayArea

Prototype

M_Int32 MC_mdaEnableOEMDisplayArea(M_Int32 screenID, M_Int32 x, M_Int32 y, M_Int32 w, M_Int32 h)

Description

This function sets the update-preventive area on a specific LCD. When playing video media through a media handler, the set area is not updated by the device playing video media.

Parameters

[in]	screened	Main LCD in case of 0, external sub LCD in case of 1
[in]	x	X coordinate
[in]	y	Y coordinate
[in]	w	Horizontal size
[in]	h	Vertical size

Return Value

Success	RegionID of the area
Failure	
M_E_INUSE	Issued when a specified area is already assigned
M_E_NOTSUP	Issued when the function is unsupported
M_E_ERROR	Issued in case of failure due to an unknown reason
M_E_INVALID	Issued in case of invalid parameter

Side Effects

None

Reference Item

None

● MC_mdaDisableOEMDisplayArea

Prototype

M_Int32 MC_mdaDisableOEMDisplayArea(M_Int32 regionID)

Description

This function releases the set update-preventive area.

Parameters

[in] regionID ID of the area

Return Values

Success		
	0	
Failure		
	M_E_NOTEXIT	Issued in case of invalid or unregistered ID
	M_E_NOTSUP	Issued when the function is unsupported
	M_E_ERROR	Issued in case of failure due to an unknown reason
	M_E_INVALID	Issued in case of invalid parameter

Side Effects

None

Reference Item

None

2.12. Placing A Call

This is a collection of APIs related to placing a call. When placing a call, a string is passed to call MC_phnCallPlace().

- **MC_phnCallPlace**

Prototype

M_Int32 MC_phnCallPlace(M_Byte* phonenumber)

Description

Places a call

Parameters

`phonenumber` Telephone number string (to be ended by null)

Return Value

 Pass

 0

 Fail

 M_E_ERROR

 Issued when a call cannot be placed

Side Effects

 None

Reference Item

 None

2.13. SMS

- MC_smsSend

Prototype

M_Int32 MC_smsSend(M_Char telnum, M_Byte* smsMsg, M_Int32 len)*

Description

This function sends a common short message. smsMsg includes the content to be included in the user data buffer of the SMS message format. The user data cannot exceed the number of bytes corresponding to the return value of MC_smsGetMaxMsgLength(). M_E_WOULDBLOCK means that data cannot be transmitted immediately due to an internal factor of the system. For the handling result of subsequent transmission of SMS messages, the events can be received according to the order that they are called through handleCletEvent.

MC_SMSEV_SEND_NOTIFY is transmitted to param1 of handleCletEvent, and 1 (number of transmitted messages) or -1 (in case of failure), to param 2. In case several events are sent, all the events will be transmitted as they are called in sequence.

Parameters

[in]	telnum	Telephone number of the party being contacted
[in]	smsMsg	String to be transmitted
[in]	len	Length of the string

Return Value

Pass	0	
Fail	M_E_INVALID	Issued when the buffer size is invalid
	M_E_WOULDBLOCK	Issued when the data cannot be transmitted immediately due to an internal factor of the system
	M_E_ERROR	Issued in case of other errors

Side Effects

None

Reference Item

None

● MC_smsGetMaxMsgLength

Prototype

M_Int32 MC_smsGetMaxMsgLength(void)

Description

Converts the maximum length of message that can be transmitted to number of bytes before it is returned

Parameters

None

Return Value

Pass

Maximum length of an SMS message in byte

Fail

M_E_ERROR

Issued in case of failure due to unknown reasons

Side Effects

None

Reference Item

None

2.14. Location Information API

Location information APIs are APIs containing both GPS information and location information of a base station.

2.14.1. API for Location Information of a Base Station

An API that returns the location information of a base station with which a terminal is communicating is provided.

● MC_lbsGetStationLocationInfo**Prototype**

```
M_Int32 MC_lbsGetStationLocationInfo(M_Int32 *baseID, M_Int32 *baseLat,  
M_Int32 *baseLong)Description
```

An API that requests the location information of a base station with which a terminal is communicating

Parameters

[out]	baseID	Base station ID
[out]	baseLat	Latitude information of the base station
[out]	baseLong	Longitude information of the base station

Return Value

Pass	0
Fail	M_E_ERROR

Side Effects

None

Reference Item

None

2.14.2. GPS Location Information API

GPS-related APIs are prepared for terminals of gpsOne solution of Qualcomm instead of GSM line or common GPS devices. Through gpsOne, location information can be received by accessing a gpsOne-related server prepared by a mobile communication service provider using the IS801-1 protocol. The C API in this section is also prepared for such scenario.

● Global structure

For GPS API, the structure should be defined as follows:

```
typedef struct MC_gpsConfig {
    M_Uint8 mode;           //Operation mode
    M_Uint8 qos;           //Quality level of location information
    M_Uint16 transport;     //Transport layer of location information
    M_Uint32 pde_addr;      // PDE (Position Determination Entity; transmission
                           // server for location information and relevant data) address
    M_Uint16 pde_port;      // PDE port
} MC_gpsConfig;

typedef enum MC_gpsMode {   // gpsOne operation mode
    MC_MS_ASSISTED = 0x1,   // MS_ASSISTED mode
    MC_MS_BASED,           // MS_BASED mode
    MC_OPT_SPEED,          // Speed optimization mode
    MC_OPT_ACCURACY        // Accuracy optimization mode
}

typedef enum MC_gpsTransport {
    SERVER_TCPIP = 0x1,
    SERVER_DBURST = 0x2;
}

typedef struct MC_locationInfo {
    M_Int32 latitude;       //latitude
    M_Int32 longitude;      // longitude
    M_Int16 altitude;       // altitude
    M_Uint16 heading;       // heading direction
}
```

```
M_Uint16 velocityHor;      // horizontal velocity
M_Int8 velocityVer;        // vertical velocity
M_Int8 accuracy;           // data accuracy
M_Char* timeString;        // time string(hhmmss)

} MC_locationInfo;

typedef enum MC_gpsCfgMask {
    MC_GPSCFG_MODE=0x1,      // Apply operation mode for gpsOne™
operation.
    MC_GPSCFG_QOS=0x2,       // Apply gpsOne™ quality level.
    MC_GPSCFG_TRANSPORT=0x4, // Apply gpsOne™ information
                             // transmission layer.
    MC_GPSCFG_SVRADDR=0x8,   // Apply PDE server address value.
    MC_GPSCFG_SVRPORT=0x10   // Apply PDE access port value.
    MC_GPSCFG_ALL=0xFFFF,    // Set configuration information.
} MC_gpsCfgMask;

typedef enum MC_gpsResult { // Result value for MC_gpsRequestLocationInfo
                             request
    MC_GPS_SUCCESS=0x01,    // GPS information reception successful
    MC_GPS_FAILED,          // GPS information reception failed
    MC_GPS_BEGIN,           // GPS starts without error on request
    MC_GPS_NO_CON,          // Not connected to the PDE server
    MC_GPS_INPROGRESS,      // GPS device already on use
    MC_GPS_LOCKED           // GPS set to Lock
} MC_gpsResult
```

● MC_gpsAvailable

Prototype

M_Int32 MC_gpsAvailable(void)

Description

Checks whether the gpsOne device is available

Parameters

None

Return Value

Pass	0	Issued when the device is available
Fail	_E_ERROR	Issued in the absence of the device

Side Effects

None

Reference Item

None

● MC_gpsRequestLocationInfo

Prototype

M_Int32 MC_gpsRequestLocationInfo(GPSREQUESTCB cb, M_Int32 interval)

Description

This is an API requesting location information by gpsOne. It operates asynchronously, and the result is reported with the event. When receiving the event, the callback function registered on the function is called.

Parameters

[in]	cb	Callback function called when receiving the GPS event
[in]	interval	Transmission is stopped when the value is -1; requests once when the value is 0, and reports every interval second (if possible) when the value is larger than 1

Return Value

Pass

Fail

M_E_INVALID	Issued when the parameter is invalid
M_E_INUSE	Issued when currently in use
M_E_ERROR	Issued for failure due to other reasons

Side Effects

None

Reference Item

None

- **GPSREQUESTCB**

Prototype

typedef void (*GPSREQUESTCB)(MC_locationInfo* pInfo, M_Int32 result)

Description

This is a callback function registered on MC_gpsRequestLocationInfo.

Parameters

[in]	pInfo	Location information by the MC_locationInfo structure; success (MC_GPS_SUCCESS) or NULL is returned
[in]	result	Result value on location request (one of the MC_gpsResult values)

Return Value

None

Side Effects

None

Reference Item

MC_gpsRequestLocationInfo

- **MC_gpsGetConfig**

Prototype

M_Int32 MC_gpsGetConfig(MC_gpsConfig *config)

Description

Gets the configuration information of gpsOne™

Parameters

[out] config: Pointer of the structure that will receive the gpsOne™ configuration information of a terminal

Return Value

Pass

0

Fail

M_E_ERROR

Side Effects

None

Reference Item

None

● MC_gpsSetConfig

Prototype

*M_Int32 MC_gpsSetConfig(MC_gpsConfig *config, MC_gpsCfgMask mask)*

Description

Sets the gpsOne™ configuration information

Parameters

- [in] config: Pointer of the structure containing configuration information to be set
- [in] mask: Target item mask among the contents of the structure

Return Value

Pass	0	
Fail	M_E_ERROR	Issued in case setting fails
	M_E_INVALID	Issued when the set data is invalid

Side Effects

None

Reference Item

None

- **MC_gpsControl**

Prototype

M_Int32 MC_gpsControl(M_Int32 function, M_Int32 command, void* argument)

Description

An API that is prepared for future expansion; since no function is defined in the specification of the current version, an error or M_E_ERROR should be returned without any condition

Parameters

[in]	function:	Type of expanded function
[in]	command:	Type of command for expanded function
[in]	argument:	Pointer for the structure of an argument for the command

Return Value

Pass	0
Fail	M_E_ERROR

Side Effects

None

Reference Item

None

2.15. Security Communication

2.15.1. Type of Related Data

This section describes the functions on the SSL library provided by WIPI to provide the SSL protocol function.

A name with data type consists of a capital letter, an under bar (_), and several numbers. A data type that does not expose the interior among data types starts with H, representing a handle; and does not use the under bar.

In SSL, most of the data are treated in the form of a handle rather than a structure, i.e., the data is used through a function provided by SSL API rather than directly.

- **HSSL**

Data representing an SSL handle; an SSL handle represents SSL on memory

- **SSL_VERSION**

Represents an SSL protocol version

```
enum SSL_VERSION_E{
    SSL_VER_SSLv3 =0x0300,
    SSL_VER_TLSv1 =0x0301};
typedef enum SSL_VERSION_E SSL_VERSION;
```

Values

SSL_VER_SSLv3 – Protocol SSLv3

SSL_VER_TLSv1 – Protocol TLSv1

- **SSL_CIPHERSUITE**

Represents a cipher suite code of SSL

```
enum SSL_CIPHERSUITE_E{
    SSL_CS_RSA_DES_192_CBC3_SHA =    0x0300000A,
```

```
SSL_CS_RSA_SEED_CBC_MD5 =      0x0300ff01,  
SSL_CS_RSA_SEED_CBC_SHA =      0x0300ff02};  
typedef enum SSL_CIPHERSUITE_E  SSL_CIPHERSUITE;
```

Values

SSL_CS_RSA_DES_192_CBC3_SHA – CipherSuite SSL3 RSA with 3DES EDE CBC SHA

SSL_CS_RSA_SEED_CBC_MD5 – CipherSuite SSL3 RSA with SEED CBC MD5

SSL_CS_RSA_SEED_CBC_SHA – CipherSuite SSL3 RSA with SEED CBC SHA

● Return Value

M_E_SSL_OK	Operation successful
M_E_SSL_OUTOFMEMORY	Insufficient memory
M_E_SSL_INVALIDARG	One or more invalid factors
M_E_SSL_POINTER	Invalid pointer
M_E_SSL_HANDLE	Invalid handle
M_E_SSL_ABORT	Progress canceled
M_E_SSL_FAIL	Progress failed
M_E_SSL_WOULDBLOCK	Blocking during SSL communication
M_E_SSL_CERTIFICATE	Invalid certificate during SSL handshaking

2.15.2. Related API

SSL API enables a WIPI program to use the SSL protocol. This API supports SSLv3 and TLSv1 protocols and certificate-handling function.

● SSLCONNECTCB

Prototype

```
void (*SSLCONNECTCB)(HSSL hSSL, M_Int32 nError, void* pParam);
```

Description

Callback function that indicates the result of SSL connection (MC_secSSLConnect) and SSL shutdown (MC_secSSLShutdown)

Parameters

[in]	hSSL	SSL handle
[in]	nError	M_E_SSL_OK: SSL connection is successfully completed. M_E_SSL_E_CERTIFICATE: The certificate used for SSLconnection is Valid. M_E_SSL_CLOSED: SSL connection is completed. M_E_SSL_FAIL: SSL connection failed.
[in]	pParam	Value inputted when calling sec_SSLConnect()

Side Effects

None

Reference Item

HSSL, MC_secSSLConnect, SSL_CIPHERSUITE

● SSLWRITECB

Prototype

void (*SSLWRITECB)(HSSL hSSL, M_Int32 nError, void* pParam);

Description

Callback function that is called when SSL Write is enabled in case M_E_SSL_E_WOULDBLOCK is returned, since MC_secSSLWrite() cannot implement SSL Write immediately

Parameters

[in]	hSSL	SSL handle
[in]	nError	M_E_SSL_OK: SSL Write is possible. M_E_SSL_FAIL: SSL Write failed.
[in]	pParam	Value inputted when calling MC_secSSLWrite()

Side Effects

None

Reference Item

HSSL, MC_secSSLWrite, SSL_CIPHERSUITE

● SSLREADCB

Prototype

void (*SSLREADCB)(HSSL hSSL, M_Int32 nError, void* pParam);

Description

Callback function that is called when SSL Read is enabled in case M_E_SSL_E_WOULDBLOCK is returned, since MC_secSSLRead() cannot implement SSL Read immediately

Parameters

[in]	hSSL	SSL handle
[in]	nError	M_E_SSL_OK: SSL Read is possible. M_E_SSL_FAIL: SSL Read failed.
[in]	pParam	Value inputted when calling MC_secSSLRead()

Side Effects

None

Reference Item

HSSL, MC_secSSLRead, SSL_CIPHERSUITE

● MC_secSSLNew**Prototype*****HSSL MC_secSSLNew(void);*****Description**

This function initializes and returns an SSL handle.

A handle is initialized to use the SSL protocol using arguments inputted by the user. This function should be initialized before using the SSL library.

The SSL handle returned by this function is necessary to execute all SSL API functions.

Parameters

None

Return Value

Pass

SSL handle

Fail

Null

Side Effects

None

Reference Item

MC_secSSLFree, HSSL, SSL_CIPHERSUITE

- **MC_secSSLFree**

Prototype

```
void MC_secSSLFree(HSSL hSSL);
```

Description

Removes the SSL handle

Parameters

[in] hSSL SSL handle

Return Value

None

Side Effects

None

Reference Item

HSSL, MC_secSSLNew, SSL_CIPHERSUITE

● MC_secSSLConnect

Prototype

```
M_Int32 MC_secSSLConnect(HSSL hSSL, M_Int32 nSocket, M_Char* szCN,
SSLCONNECTCB cbSSLConnect, void* pParam);
```

Description

Attempts SSL connection to a specified server using a given SSL handle

Parameters

[in]	hSSL	SSL handle
[in]	nSocket	Socket handle that succeeded in connection
[in]	szCN	Server address needed for server authentication
[in]	cbSSLConnect	Callback function that is called in case of successful or failed connection attempt
[in]	pParam	Value to be transmitted when a callback function is called

Return Value

Pass

M_E_SSL_OK

Fail

M_E_SSL_HANDLE

Issued when the SSL handle is invalid

M_E_SSL_INVALIDARG

Issued when the socket handle is invalid

M_E_SSL_OUTOFMEMORY

Issued in case of insufficient memory

M_E_SSL_WOULDBLOCK

SSL connection (handshake process) in progress (the result will be transmitted through the callback function)

Side Effects

None

Reference Item

HSSL, SSLCONNECTCB, SSL_CIPHERSUITE

● **MC_secSSLAddCipherSuite**

Prototype

M_Int32 MC_secSSLAddCipherSuite(HSSL hSSL, SSL_CIPHERSUITE nCipherSuite);

Description

This function adds a cipher suite to the client-side cipher suite list to be used in a handshake for implementation for SSL communication.

Required for executing MC_secSSLConnect, Cipher suite is called before attempting MC_secSSLConnect to add a cipher suite. Up to three cipher suites can be added.

Parameters

[in] hSSL SSL handle
[in] nCipherSuite Cipher suite to be added

Return Value

Pass	M_E_SSL_OK	
Fail	M_E_SSL_HANDLE	Issued when the SSL handle is invalid
	M_E_SSL_INVALIDARG	Issued when the cipher suite is invalid
	M_E_SSL_FAIL	Issued when SSL connection is already made or is being terminated, can no longer be added, or already added

Side Effects

None

Reference Item

HSSL, SSL_CIPHERSUITE, MC_secSSLClearCipherSuite

● MC_secSSLClearCipherSuite

Prototype

M_Int32 MC_secSSLClearCipherSuite(HSSL hSSL);

Description

Clears an added cipher suite list of MC_secSSLAddCipherSuite added

Parameters

None

Return Value

Pass

M_E_SSL_OK

Fail

M_E_SSL_HANDLE

Issued when the SSL handle is invalid

M_E_SSL_FAIL

Issued when SSL connection is
already made or is being terminated

Side Effects

None

Reference Item

HSSL, MC_secSSLAddCipherSuite, SSL_CIPHERSUITE

● MC_secSSLWrite

Prototype

M_Int32 MC_secSSLWrite(HSSL hSSL, M_Uint8* pData, M_Int32 nSize, SSLWRITECB cbSSLWrite, void* pParam);

Description

Attempts to transmit as much data as specified through a connected channel using the SSL handle

Parameters

[in]	hSSL	SSL handle
[in]	pData	Data to be transmitted
[in]	nSize	Size of data to be transmitted
[in]	cbSSLWrite	Callback function that will report the result of data transmission during socket blocking and return SSL_E_WOULDBLOCK
[in]	pParam	Value to be transmitted when a callback function is called

Return Value

Return Value

Pass

Returns the data length transmitted

Fail

M_E_SSL_HANDLE	Issued when the SSL handle is invalid
M_E_SSL_HANDLE	Issued when the SSL handle is invalid
M_E_SSL_INVALIDARG	Issued when the buffer or buffer size is invalid
M_E_SSL_OUTOFMEMORY	Issued in case of insufficient memory
M_E_SSL_WOULDBLOCK	Issued when SSL cannot transmit data immediately (the result will be indicated through the callback function)
M_E_SSL_FAIL	Issued when SSL Write failed

Side Effects

None

Reference Item

HSSL, SSLREADCB, SSL_CIPHERSUITE

● MC_secSSLRead()

Prototype

```
M_Int32 MC_secSSLRead(HSSL hSSL, M_Uint8* pData, M_Int32 nSize,
SSLREADCB cbSSLRead, void* pParam);
```

Description

Attempts to read as much data as what is given through a connected channel using a given SSL handle

Parameters

[in]	hSSL	SSL handle
[out]	pData	Buffer where the data read is to be stored
[in]	nSize	Size of the data to be read
[in]	cbSSLRead	Callback function that will indicate the result of data read during socket blocking and return SSL_E_WOULDBLOCK
[in]	pParam	Value to be transmitted when a callback function is called

Return Value

Pass

Returns the data length transmitted

Fail

M_E_SSL_HANDLE	Issued when the SSL handle is invalid
M_E_SSL_INVALIDARG	Issued when the buffer or buffer size is invalid
M_E_SSL_OUTOFMEMORY	Issued in case of insufficient memory
M_E_SSL_WOULDBLOCK	Issued when SSL cannot transmit data immediately (the result will be indicated through the callback function)
M_E_SSL_FAIL	Issued when SSL Write failed

Side Effects

None

Reference Item

HSSL, SSLREADCB, SSL_CIPHERSUITE

● MC_secSSLInstallCert**Prototype*****M_Int32 MC_secSSLInstallCert(HSSL hSSL);*****Description**

Stores the Server Certificate (Peer Certificate) in a repository

Parameters

[in]	hSSL	SSL handle
------	------	------------

Return Value

Pass

M_E_SSL_OK

Fail

M_E_SSL_HANDLE

Issued when the SSL handle is invalid

M_E_SSL_OUTOFMEMORY

Issued in case of insufficient memory

M_E_SSL_FAIL

Issued when the certificate installation failed

Side Effects

None

Reference Item

HSSL, SSL_CIPHERSUITE, MC_secSSLRemoveCertAll

● MC_secSSLRemoveAll**Prototype*****M_Int32 MC_secSSLRemoveCertAll(HSSL hSSL);*****Description**

Removes all certificates stored in a certificate repository.

Parameters

[in]	hSSL	SSL handle
------	------	------------

Return Value

Pass

M_E_SSL_OK

Fail

M_E_SSL_HANDLE

Issued when the SSL handle is invalid

M_E_SSL_FAIL

Issued when certificate removal failed

Side Effects

None

Reference Item

HSSL, SSL_CIPHERSUITE, MC_secSSLInstallCert

● **MC_secSSLContinue**

Prototype

M_Int32 MC_secSSLContinue(HSSL hSSL);

Description

When a Server Certificate cannot be trusted during the implementation of SSL connection, the connection is terminated. In such cases, this function reactivates the terminated SSL connection.

Parameters

[in] hSSL SSL handle

Return Value

Pass	M_E_SSL_OK	
Fail	M_E_SSL_HANDLE	Issued when the SSL handle is invalid
	M_E_SSL_FAIL	Issued when the resumption of the terminated SSL connection failed

Side Effects

None

Reference Item

HSSL, SSL_CIPHERSUITE

- **MC_secSSLVer**

Prototype

M_Int32 MC_secSSLVer();

Description

gets TLS library version information.

Parameters

None

Return Value.

Pass

TLS library version

Fail

M_E_SSL_FAIL

Failed SSL library version information.

Side Effects

None.

Reference Item

HSSL, SSL_CIPHERSUITE

● MC_secSSLShutdown

Prototype

M_Int32 MC_secSSLShutdown(HSSL hSSL, M_Int32 nReason);

Description

Closes the connection using a given SSL handle

Parameters

[in]	hSSL	SSL handle
[in]	nReason	Reason of closing connection (Enter -1 for closing external network, 0 for normal closing)

Return Value

Pass

M_E_SSL_OK.

Fail

M_E_SSL_HANDLE

Issued when the SSL handle is invalid

M_E_SSL_FAIL

Issued when SSL connection termination failed

M_E_SSL_WOULDBLOCK

Issued when SSL connection cannot be terminated immediately (the result will be indicated through the callback function set in sec_SSLConnect)

Side Effects

None

Reference Item

HSSL, SSL_CIPHERSUITE, SSLCONNECTCB

2.16. Control of Supplementary Devices

Functions that are used to control supplementary devices supported by the terminal

- **MC_BackLight**

Prototype

```
typedef enum _MC_BackLight MC_BackLight {  
    MC_LIGHT_ON = 0                // Turn on the backlight.  
    MC_LIGHT_OFF,                  // Turn off the backlight.  
    MC_LIGHT_ALWAYS_ON,           // Keep the backlight on.  
    MC_LIGHT_DEFAULT,             // Maintain the user setting.  
} MC_BackLight;
```

Description

Enumeration type representing backlight control options

● MC_miscBackLight

Prototype

M_Int32 MC_miscBackLight(M_Int32 id, MC_BackLight on_off, M_Int32 color, M_Int32 timeout)

Description

This is a function that controls the backlight. In a terminal, a backlight setting is set as a default value. This value maintains the same state for a few seconds when a key is pressed. This function is designed to invalidate the value; to return to the original state after it is used by the application, reset the value using MC_LIGHT_DEFAULT before closing. In case of timeout, the backlight goes off automatically. Backlight number 0 represents the backlight of the main LCD, and backlight number 1, the backlight of the auxiliary LCD. When a color can be specified in a backlight, parameter color whose value takes the 0xYYRRGGBB (network byte ordering) format is used. Here, YY is ignored, and RR, GG, and BB specify the range of red, green, and blue, respectively.

Parameters

id	Backlight number
on_off	Backlight adjustment option
color	Backlight color
timeout	Timer value in millisecond; referred to by on_off only in case of MC_DEV_LIGHT_ON

Return Value

Pass	0
Fail	M_E_ERROR –Issued in the absence of a specified backlight

Side Effects

None

Reference Item

None

● MC_miscSetLed

Prototype

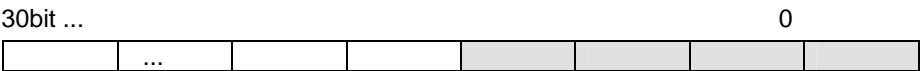
M_Int32 MC_miscSetLed (M_Int32 leds)

Description

Sets On/Off for LEDs; when each bit is 1, it represents On (when each bit is 0, it represents Off)

Example:

When there are 4 external LEDs, 4 bits are used beginning with LSB bit.



MC_miscSetLed(0x3) - Turns on two LEDs, turns off two LEDs

In case of a terminal that supports color, however, the brightest color set by the manufacturer is set.

Parameters

[in] leds When the bit is 1, the corresponding LED is turned on; when the bit is 0, the corresponding LED is turned off.

Return Value

Pass

bit OR value of On/Off state set for each LED (≥ 0)

Fail

M_E_INUSE Issued when the currently requested LED is in use

M_E_ERROR Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

Prototype

Description

Sets On/Off for LEDs

Example:

When there are 4 external LEDs, 4 bits are used beginning with LSB bit.



MC_miscSetLedControl(0x3) - Turns on two LEDs, turns off two LEDs

In case of a terminal that supports color, however, the brightest color set by the manufacturer is set.

Parameters

[in] leds led to control

[in] mask mask bit

Return Value

Pass

bit OR value of On/Off state set for each LED (≥ 0)

Fail

M_E_INUSE	Issued when the currently requested LED is in use
-----------	---

M E ERROR	Issued in case of failure due to other reasons
-----------	--

Side Effects

None

Reference Item

None

- **MC_miscGetLed**

Prototype

M_Int32 MC_miscGetLed (void)

Description

Reads the On/Off state of LED

Parameters

None

Return Value

Bit OR value of On/Off state for each LED

Side Effects

None

Reference Item

None

- **MC_miscGetLedCount**

Prototype

M_Int32 MC_miscGetLedCount (void)

Description

Gets the number of external LEDs

Return Value

Number of LEDs

Side Effects

None

Reference Item

None

● MC_miscGetLedSupportColor**Prototype**

M_Int32 MC_miscGetLedSupportColor (M_Int32 led, M_Int32 RGB[])

Description

Brings a color list provided by the relevant LED of a terminal

Parameters

[in]	led	LED to be controlled
[out]	RGB	Color list

Return Value

Number of colors provided

Side Effects

None

Reference Item

None

- **MC_miscGetLedColor**

Prototype

M_Int32 MC_miscGetLedCount (M_Int32 led)

Description

Brings the currently specified RGB value for the relevant LED

Parameters

[in]	led	LED to be controlled
------	-----	----------------------

Return Value

RGB value

Side Effects

None

Reference Item

None

● MC_miscSetLedColor

Prototype

M_Int32 MC_miscSetLedColor (M_Int32 led, M_Int32 RGB)

Description

Specifies the RGB value to the relevant LED of a terminal; in the absence of an RGB value received as an argument, however, the approximate value supported is set

Parameters

[in]	led	LED to be controlled
[in]	RGB	RGB value

Return Value

Pass	RGB value set
Fail	
	M_E_INUSE Issued when the currently requested LED is in use
	M_E_ERROR Issued in case of failure due to other reasons

Side Effects

None

Reference Item

None

2.17. Mathematical Operation

● MC_mathSin100**Prototype*****M_Int32 MC_mathSin100(M_Int32 angle)*****Description**

Gets the Sin value for a specified angle and returns an integer value that is 100 times the result

Parameters

angle Unit value of degree

Return Value

Pass

An integer value that is 100 times the Sin value for a specified angle

Fail

None

Side Effects

None

Reference Item

None

● MC_mathCos100**Prototype**

M_Int32 MC_mathCos100(M_Int32 angle)

Description

Gets the Cos value for a specified angle and returns an integer value that is 100 times the result

Parameters

angle Unit value of degree

Return Value

Pass

An integer value that is 100 times the Cos value for a specified angle

Fail

None

Side Effects

None

Reference Item

None

- **MC_mathTan100**

Prototype

M_Int32 MC_mathTan100(M_Int32 angle)

Description

Gets the Tan value for a specified angle and returns an integer value that is 100 times the result

Parameters

angle Unit value of degree

Return Value

Pass

An integer value that is 100 times the Tan value for a specified angle

Fail

None

Side Effects

None

Reference Item

None

● MC_mathArcSin100**Prototype*****M_Int32 MC_mathArcSin100(M_Int32 arc)*****Description**

Returns the angle of asin in integer value

Parameters

arc 100 times the Sin value for an angle between -100 to 100

Return Value

Pass

Sin value for an angle between -90 and 90

Fail

-32767

Side Effects

None

Reference Item

None

● MC_mathArcCos100**Prototype**

M_Int32 MC_mathArcCos100(M_Int32 arc)

Description

Returns the angle of acos in integer value

Parameters

arc 100 times the Cos value for an angle between -100 to 100

Return Value

Pass

Cos value for an angle between -90 and 90

Fail

-32767

Side Effects

None

Reference Item

None

● MC_mathArcTan100**Prototype**

M_Int32 MC_mathArcTan100(M_Int32 arc)

Description

Returns the angle of acos in integer value

Parameters

arc 100 times the Tan value for an angle between -100 to 100

Return Value

Pass

Tan value for an angle between -90 and 90

Fail

-32767

Side Effects

None

Reference Item

None

- **MC_mathAbs**

Prototype

M_Int32 MC_mathAbs(M_Int32 a)

Description

Returns the absolute value in integer

Parameters

a Integer value for which an absolute value is sought

Return Value

Pass

Absolute integer value

Fail

None

Side Effects

None

Reference Item

None

● MC_mathMaxN**Prototype*****M_Int32 MC_mathMaxN(M_Int32 *a, M_Int32 n)*****Description**

Compares n number of integer values and returns the biggest value

Parameters

- | | |
|---|--|
| a | Address of a repository for consecutive integer values |
| n | Number of integers for comparison |

Return Value

- | | |
|------|--|
| Pass | |
| | Biggest value among n number of integers |
| Fail | |
| | None |

Side Effects

- | |
|------|
| None |
|------|

Reference Item

- | |
|------|
| None |
|------|

● MC_mathMinN

Prototype

*M_Int32 MC_mathMinN(M_Int32 *a, M_Int32 n)*

Description

Compares //n number (n개) of integer values and returns the smallest value

Parameters

- | | |
|---|--|
| a | Address of a repository for consecutive integer values |
| n | Number of integers for comparison |

Return Value

- | | |
|------|---|
| Pass | |
| | Smallest value among n number of integers |
| Fail | |
| | None |

Side Effects

None

Reference Item

None

● MC_mathCheck**Prototype**

M_Boolean MC_mathCheck(M_Int32 a, M_Int32 b, M_Int32 c)

Description

Checks whether a specified integer value is within a specified range and returns the result

Parameters

- a Beginning value of the check
- b Ending value of the check
- c Value to be checked, i.e., whether it is within the range

Return Value

- Pass
TRUE (when the integer is between two values)
- Fail
FALSE (when the integer is outside two values)

Side Effects

None

Reference Item

None

● MC_mathRand**Prototype**

M_Int32 MC_mathRand(M_Int32 lower, M_Int32 upper)

Description

Returns a random number within the range of a specified integer value

Parameters

lower lower bound

upper upper bound

Return Value

Pass

Random number generated

Fail

None

Side Effects

None

Reference Item

None

- **MC_mathSrand**

Prototype

M_Int32 MC_mathSrand (M_Int32 seed)

Description

Changes a random seed

Parameters

seed Random number seed value

Return Value

None

Side Effects

None

Reference Item

None

3. Standard C Library Function

For the convenience of C language developers, the following C libraries should be supported (the supported functions will be supported in the same manner as ANSI-C interfaces):

- **String-Related Functions (string.h)**

strcpy, strncpy, strcat, strncat, strcmp, strncmp, strchr, strrchr, strspn, strcspn, strpbrk, strstr, strlen, strtok, memcpy, memmove, memcmp, memchr, memset

- **Standard Library Functions (stdlib.h)**

atof, atoi, atoll, strtod, strtol, strtoul

- **Variable Parameter-Related Functions (stdarg.h)**

va_list, va_start, va_arg, va_end

- **Time-Related Functions (time.h)**

clock, time, difftime, mktime, localtime, gmtime